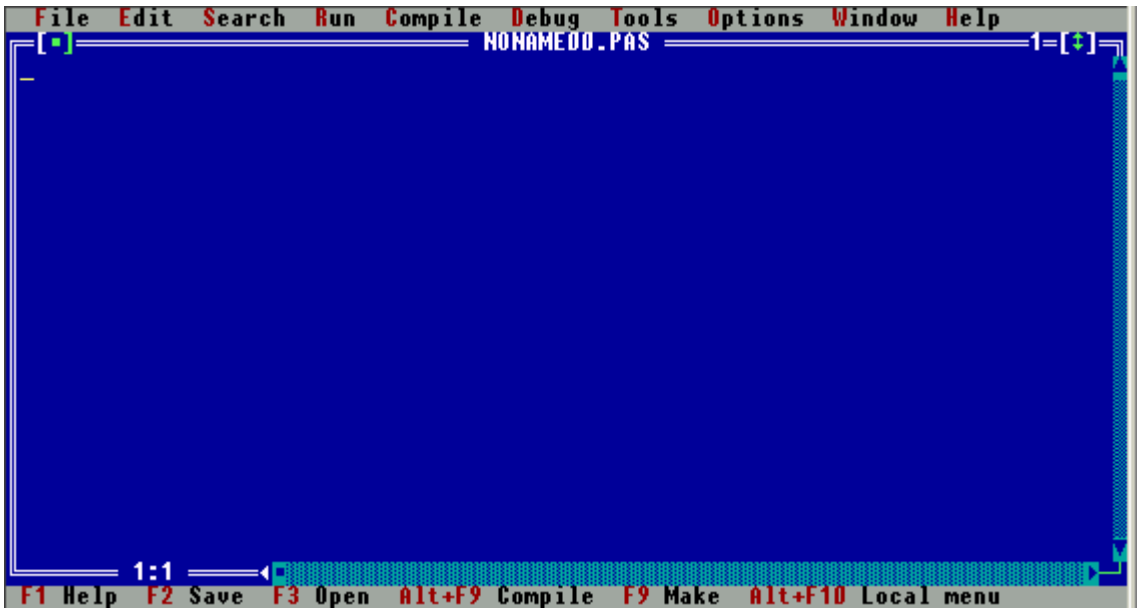
Pascal 语言教程

qiyuanwj 整理

Pascal 第一讲:编程环境

一、Pascal 的启动

装有 Turbo Pascal 的目录中，双击 turbo.exe。在桌面上或程序组中双击 turbo 图标。



常用命令的快捷方式

F3：打开文件

Alt+F3：关闭一个文件

F2：存储

F6：切换一个窗口

Ctrl+Insert：复制

Shift+Del：剪切

Shift+Insert：粘贴

Ctrl+Y：删除一行

Alt+Backspace：撤消

一个 PASCAL 程序

进入编辑状态：

选择菜单 [FILE][OPEN]（今后涉及到菜单调用时，均以此格式表示，即选择 FILE 菜单中的 OPEN 功能）功能，打开一个程序文件，在输入文件名时，输入文件的主名即可进入程序编辑状态。这时可用编辑功能健：

光标键：用以上下左右移动光标位置；

INSERT：插入 /改写状态切换；

TAB；光标跳至下一制表位；

CAPSLOCK：大小写切换；

DELETE：删除光标位一字符；

BACKSPACE：删除光标前一字符；

CTRL+Y：删除光标处一行；

HOME：光标跳至行首；

END：光标跳至行未；

PAGEUP：上翻一页；

PAGEDOWN：下翻一页；

ENTER：回车 /换行；

编辑一个 PASCAL 程序：

输出一句话的小程序：

```
program Q1;
begin
writeln( ‘ You are welcome to the PASCAL world ! ’ );
readln;
end.
```

请在编辑状态下编辑输入上述程序，第一行为程序开头，程序名；第二行 BEGIN 表示程序开始，最后一行 END 表示程序结束；WRITELN（）语句把括号中单引号中的的字符打印在屏幕上； READLN 语句等待用户敲回车键结束程序。记住，一般每行以分号“；”结束，但 END 后以句号“ .”结束，而 BEGIN 后没有标点符号。即：

除 BEGIN 外，每一句 PASCAL 语句后均有分号或句号，只有程序的最后一个 END 后才用句号，而其它任何行未都是分号！！！

在集成环境中运行程序：

上述程序 [例 1、1]编辑好后，如果要运行，只需选择菜单 [RUN][RUN] 命令（或 CRTL+F9），如果程序正确的，就会在屏幕（这个屏幕叫程序输出屏幕）上显示：

You are welcome to the PASCAL world !

然后我们按回车键，就能又回到集成环境中。这时如果还想查看刚才的屏幕显示，只需选择菜单 [RUN][USER SCREEN]（或 ALT+F5），就又能到输出屏幕，敲任意键又回到集成环境中。

如果程度出错，则程序没有被运行完就会回到集成环境中，并且光标停在错误的这一行，在编辑窗口中的第一行还会显示出错信息，如：

Run-time error 错误代码 at 错误发生地址

保存文件

把当前编辑的程序文件以当前名存盘，只需运行菜单命令 [FILE][SAVE]（或 F2），即可。

关闭当前文件

当前文件不想现在再编辑了，可把它关闭掉，即运行菜单命令 [WINDOWS][CLOSE]（或 ALT+F3），即可。

打开已有文件

欲打开一个已经存在的程序文件，运行菜单命令 [FILE][OPEN]（或

F3），再按 TAB 键去选择或不按 TAB 键而直接输入文件名即可。

练习

在集成环境中输入以下程序，程序的作用是计算圆的面积，圆的半径由用户从键盘输入，编辑运行正确后请存盘：

```
program area_of_cicle;
const pi=3.1416;
var s:real;
      r:integer;
begin
   writeln('Please input radius :');
   readln(r);
   s:=pi*r*r;
   writeln('s=',s);
   readln;
end.
```

运行举例：

Please input radius:

5

s=7.854000000E+01

PASCAL 第二讲:程序的基本结构

一、程序的基本结构

```
program area_of_cicle;
const pi=3.1416;
var s:real;
   r:integer;
begin
   writeln('Please input radius :');
   readln(r);
   s:=pi*r*r;
   writeln('s=',s);
   readln;
end.
```

每一个 PASCAL 程序都由程序头部和程序主体组成，最后以“ end.”作为整个程序的结束。

程序头部

程序头部毫无疑问是在程序的开头位置，以“ program”这个词开始（但经常省略这一），
以第一个 BEGIN 的前一行结束，中间每行后均有分号。

以 CONST 为开始的部分是说明程序中要用到的常量，以 VAR 为开始的部分是说明程序中要用
到的变量。即程序中要用到的所有的常量及变量，我们必需在程序首部加以说明其名称及类
型。这些我们将在稍后讲到。

程序主体

以第一个 BEGIN 开始，以最后一个 END 结束，中间即为程序命令行，每一行均以分号结束！

二、Pascal 用到的数和符号

1、PASCAL 语言的字符表

是 ASCII 字符集，主要有：

⑴26 个英文字母(不分大小写)

⑵十个数字符号

⑶特殊符号。如+-*/=><］［：；．等

2、标识符

以字母开头的字母数字序列(大小写等效，可跟下划线_)，用来标识常量、变量、程序、函数等。

自定义标识符时要注意遵循此规则。

3、标准标识符与保留字

标准标识符有 40 个，五类(false true maxint Boolean real integer char abs trunc read
write)详见书 P16。

保留字是 Pascal 语言中具有特定的含义的字符。在 PASCAL7.0 中书

写保留字的时候，字符颜色会

变为白色。

一共 36 个保留字：

(program function begin end procedure var const array if then else case fo

r to do

repeat until while and div in mod not or nil

4、常量和变量：

程序设计中经常要用到常量和变量，这些都必须先定义后才能使用。

PASCAL 语言的常量与变量都必须在程序头部先加以说明，即说明常量、变量的的名称及数据类型。

PASCAL 语言的数据类型很多，最常用的有以下几种：

整数类型（没有小数部分）

INTEGER：取值范围 [-32768，32767]。占用内存 2 个字节（ 16 位）。

WORD：取值范围 [0， 65535]。占用内存 2 个字节（ 16 位）。

BYTE：取值范围 [0， 255]。占用内存 1 个字节（ 8 位）。

LONGINT：取值范围 [-2147483648， 2147483647]。占用内存 4 个字节（ 32 位）。

SHORTINT：取值范围 [-128， 127]。占用内存 1 个字节（ 8 位）。

特点：

可进行+、-、*、DIV(整除)、MOD(取余)
注意不能用/，因为得到的结果可能不为整型。

实数类型
REAL：取值范围 [2.9E-39， 1.7E+38]。占用内存 6 个字节（ 48 位）。

实数的表示法有两种：

1、十进制表示法，如： -0.4576, 123.567, -234, 0

2、科学计数法，如： 1.234E-4, -3.546E12

表示 1.234 乘 10 的负 4 次 .方； -3.546 乘 10 的 12 次方。

特点：

包括正实数、负实数和零。运算慢，无法精确表示。

可进行+、-、*、/运算。整数与实数运算时自动转为实数进行运算。

字符类型

CHAR：单字符型，即取 1 个字符，如‘ A’，‘ 1’。

STRING：字符串型，即多个字符，如‘ ABCD！123’，‘ %￥ #DS12’。

如果一个常 /变量只要用到 1 个字符，则可把它定义成 CHAR 型；如果是多个字符，但不知

确切多少个，则定义成 STRING 型，如果知道确切有 N 个字符，则定义成 STRING[N]型。字

符串在 PASCAL 中使用都是加上单引号。

特点：

字符常 /变量是由单个字符组成的，所有字符都来自 ASCII 字符集。字符的数据是用两个单

引号把单个字符括起来表示的。

每个字符都有一个序值(在 ASCII 字符集的位置)，可以用六个关系运算符（＞，＝，＜，＞＝，

＜＝，！＝）来比较大小。

布尔型

BOOLEAN：布尔型即为逻辑型，取值为 TRUE、 FALSE， 即真、假。也有序，true 为 1，false 为 0。

布尔型有三个运算符：and or not

常量：指程序运行过程中，其值不能改变的量。 在程序头部中说明，以 CONST 引导。

分为四类：

⑴整型常量

⑵实型常量

⑶字符常量

⑷布尔型常量

格式：

const <常量标识符>=<常量>;

例：const pi:=3.14;

变量：是指在程序执行过程中，其值可以改变的量。 在程序头部中加以说明，以 VAR 引导。

变量三要素：变量名、变量类型、变量值。

⑴变量名用一个合法的标识符来表示。

⑵变量在某一固定时刻是用来存放常量的，而常量是有类型的数据，因而变量也是有类型的。

类型不能有两义性。

⑶变量值。在程序中由赋值语句来赋值。

var <变量标识符列表>:<类型>;

例：var a,b:integer; c,d:real;

5、表达式

由算术运算符（ +，—， *， /）及数字、常量、变量、标准函数所组成的式子中心表达式。

如： 5+9； A*B-34*ABS（ -34） /INT（ B）

另外， PASCAL 中有两个很有用的运算符， DIV：求商的整数值；MOD：求余数。如：

8 DIV 3 的值是 2， 10 DIV 3 的值是 3；

8 MOD 3 的值是 2， 10 MOD 3 的值是 1；

在 PASCAL 的表达式中，不允许出现我们日常生活中的那种分式或根号等式子，我们必须按照
PASCAL 的语法把它们改写成 PASCAL 表达式。

Pascal 语言中的运算符及其优先级

单目运算符 (最高优先级)
@ 取变量或函数的地址(返回一个指针)
not 逻辑取反或按位取反
乘除及按位运算符

* 相乘或集合交集

/ 浮点相除

div 整数相除

mod 取模 (整数相除的余数)

as 程序运行阶段类型转换 (RTTI 运算符)

and 逻辑或按位求和

shl 按位左移

shr 按位右移

加减运算符

+ 相加、集合并集、字符串连接或指针增加一个偏移量

- 相减、集合差集或指针减少一个偏移量

or 逻辑或按位或运算

xor 逻辑或按位异或运算

关系及比较运算符(最低优先级)

= 判断是否相等

<> 判断是否不相等

< 判断是否小于

> 判断是否大于

<= 判断是否小于或等于,或是否是一个集合的子集

>= 判断是否大于或等于,或是否是一个集合的父集

in 判断是否是集合成员

is 判断对象是否类型兼容 (又一个 RTTI 运算符)

具体优先顺序：

⑴括号内先算

⑵函数

⑶运算符优先顺序

⑷同级运算按从左到右的次序。

注意：

１、与大多数编程语言相反，Pascal 语言中 and 和 or 运算符的优先级比关系运算符高。因此，如果你
的代码为 a < b and c < d，编译器首先会编译 and 运算符，由此导致编译出错。为此你应该把每个
< 表达式用小括号括起来： (a < b) and (c < d)。
２、 同一种运算符用于不同数据类型时它的作用不同。例如，运算符 + 可以计算两个数字的和、连
接两个字符串、求两个集合的并集、甚至给 PChar 指针加一个偏移量。

然而，你不能象在 C 语言中那

样将两个字符相加。

３、另一个特殊的运算符是 div。在 Pascal 中，你能用 / 计算两个数字（实数或整数）的商，而且

你总能得到一个实型结果。如果计算两个整数的商并想要一个整型结果，那么就需要用 div 运算符。

例如：把下列算式改写成 PASCAL 表达式：

改写为 PASCAL 表达式为： (x*x+3*y-5*(z-2))/(x-y*y)

从上例中可以看出：

运算符两端，除实型和整数型外不允许为两种不同的数据类型。

PASCAL 表达式中没有分式，只能以除号“ /”来隔开；

PASCAL 表达式中的分子与分母应该用括号括开；

PASCAL 表达式中只有小括号，不能有中括号或大括号，小括号可以有很多层；

PASCAL 表达式中没有乘幂，只能用乘法来表达；

PASCAL*表达式中任意两个常量、变量、数值、括号、函数之间都必须不能缺省运算符，即乘号
必不可少；
这些要求希望大家记熟，这是我们编写 PASCAL 程序的必要基础。

表达式的数据类型根据它的值来划分。(所以表达式分为算术表达式、字符表达式、布尔表达式)

6、标准函数的使用。

PASCAL 语言提供数量非常大的标准函数供我们使用，这些函数有些可以直接调用，有些放在另外的
单元（ UNIT）中。 PASCAL 也提供了大量的标准过程，和标准函数一样供用户直接或间接调用。

如果一个函数或过程不在 SYSTEM（即默认调用的单元）中，而在其它单元中话，即在程序头部的第
一行正式行中加上 USES 单元名；如，如果要使用 CLRSCR（清屏过程，在 DOS 单元中）的话，就必
须在程序头部加上： USES DOS；

函数的调用：如： A： =ABS（ -34），即把 -34 的绝对值赋给变量

A；（注意：函数只能出现在表

达式中，即赋值语句中）。此时： A 的数据类型与括号中的参数 -34

是一致的。函数的语法中都会说

明，这个函数的参数的类型及这个函数的值（结果）的类型。

过程的调用：过程即命令，如： CLRACR； 程序运行结果是清屏。

常用标准函数与过程很多，这里只列出最常用的几个。

标准函数

函数名

语法

说明

Abs

Abs （ r:real/integer） :real/integer

返回参数 R 的绝对值，类型与参数相同

Chr

chr （ i： integer）： char

返回参数所对应的 ASCII 码值，类型为 CHAR

Copy

Copy(s:string;n,m:integer):string

返回字串 S 的第 N 个字符开始的 M 个字符

Cos

Cos(r:real):real

返回参数 R 的余弦值

Exp

Exp(r:real):real

返回参数 R 的以 e 为底的幂

Int

Int(r:real):real

返回参数 R 的整数部分，返回的值是实数类型

Length

Length(s:string):integer

返回字串 S 的长度

Ln

Ln(r:real):real

返回参数 R 的自然对数

odd

Odd(I:integer):boolean

判断参数 I 是否奇数，如是，则返回 TRUE

ord

Ord(s:scalar):integer

返回任意有序量的序值

random

Random

返回 0 至于间的任意一个小数（随机数）

round

Round(r:real):longint

返回参数 R 的四舍五入取整值

sin

Sin(r:real):real

返回参数 R 的正弦值

sqrt

sqrt(r:real):real

返回参数 R 的平方根

trunc

Trunc(r:real):integer

返回参数 R 的整数部分，返回的值是整数类型

标准过程

过程名

语法

说明

Delay (CRT 单元)

Delay(ms:word)

延迟发声 MS 毫秒

Delete

Delete(s:string;n,m:integer)

把串 S 中的第 N 个字符开始的 M 个字符删除

Exit

Exit

从当前执行的程序中退出

Gotoxy (CRT 单元)

Gotoxy(x,y:integer)

把光标定位到第 X 列 Y 行处

halt

Halt

中断程序的运行

nosound

Nosound

关闭机器喇叭

Sound (CRT 单元)

Sound(f:word)

让机器喇叭发出频率为 F 的声音，直到 nosound

str

Str(I:integer;var s:string)

把数值 I 转换为字符串 S

val

Val(s:string;var r:real;var c:word)

把字符串 S 转换为数值 R，并返回错误代码 C

练习题：

一、判断以下标识符的合法性：

a3　　3a　　a17　　abcd　　ex9.5　　α　　β　　λ

二、将下列的数学表达式改写成 PASCAL 表达式：

b^2-4ac

三、求下列表达式的值：

20 mod 19　　15 mod 9　　7 div 8　　19 div 3

(4>5)　and　(7<8)

(8>9)　or　(9<10)

2　and　((3=3)　or　(3<7))

31 div (5 mod 2)

31 div 5 mod 2

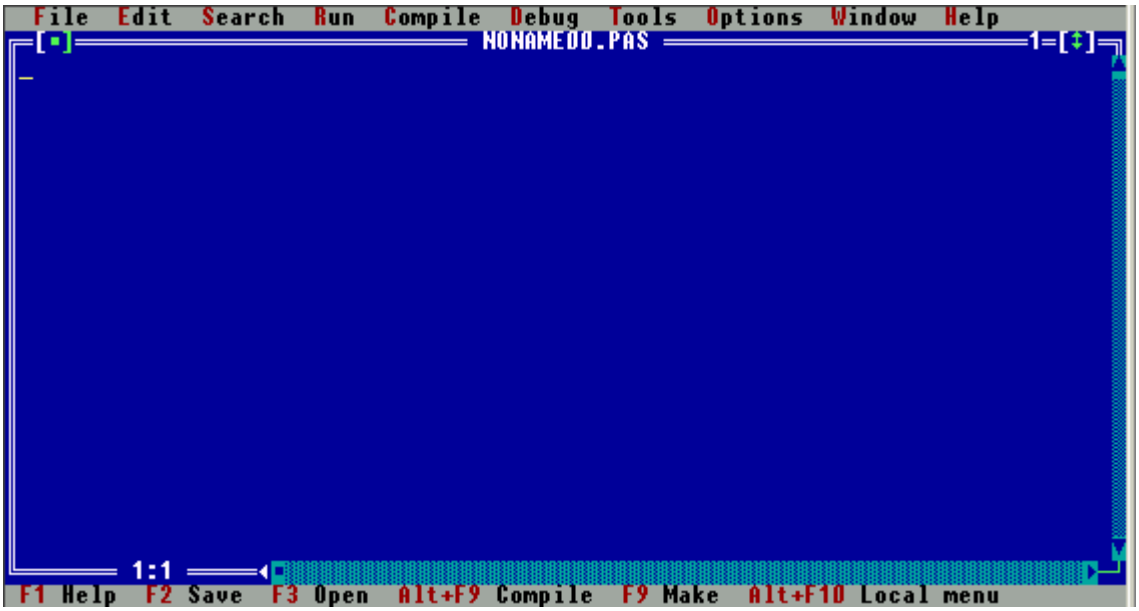
31 /5 div 2

四、把下列算式改写为 PASCAL 表达式：

$$\frac{a^3-b(c^2-abc)+4c}{\frac{1+ab^2}{c+2a}}$$

第三讲:程序的调试

一、Turbo Pascal 环境

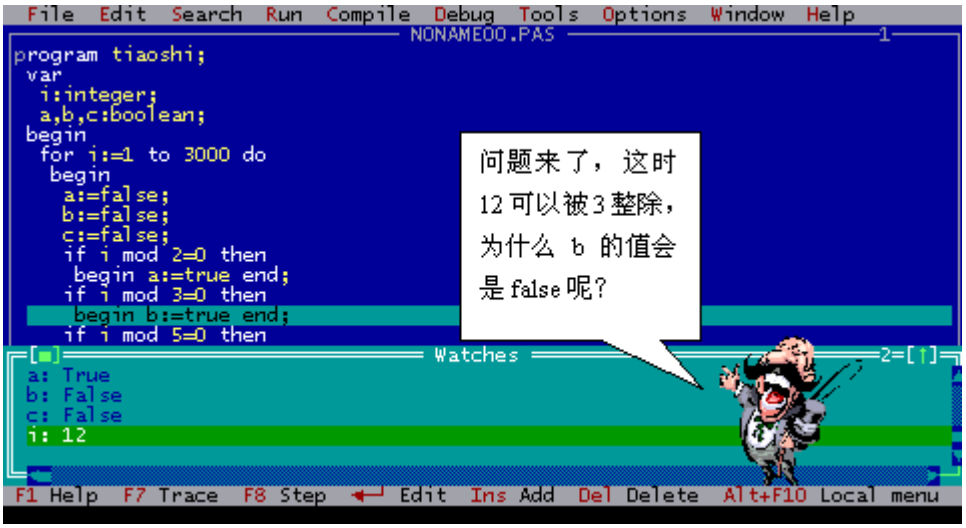


二、程序的调试（单步调试）

任何一个天才都不敢说，他编的程序是100%正确的。几乎每一个稍微复杂一点的程序都必须经过反复的调试，修改，最终才完成。所

以说，程序的调试是编程中的一项重要技术。我们现在就来掌握一下基本的程序调试。 我们以下的示范，是以时下比较流行的 Borland Pascal 7.0 为例子，其他的编程环境可能略有不同，但大致上是一致的。

我们先编一个比较简单的程序，看看程序是如何调试的。




```
program tiaoshi;
var i:integer;
begin
      for i:=1 to 300 do
      begin
            if i mod 2 = 0 then
                  if i mod 3 = 0 then
```

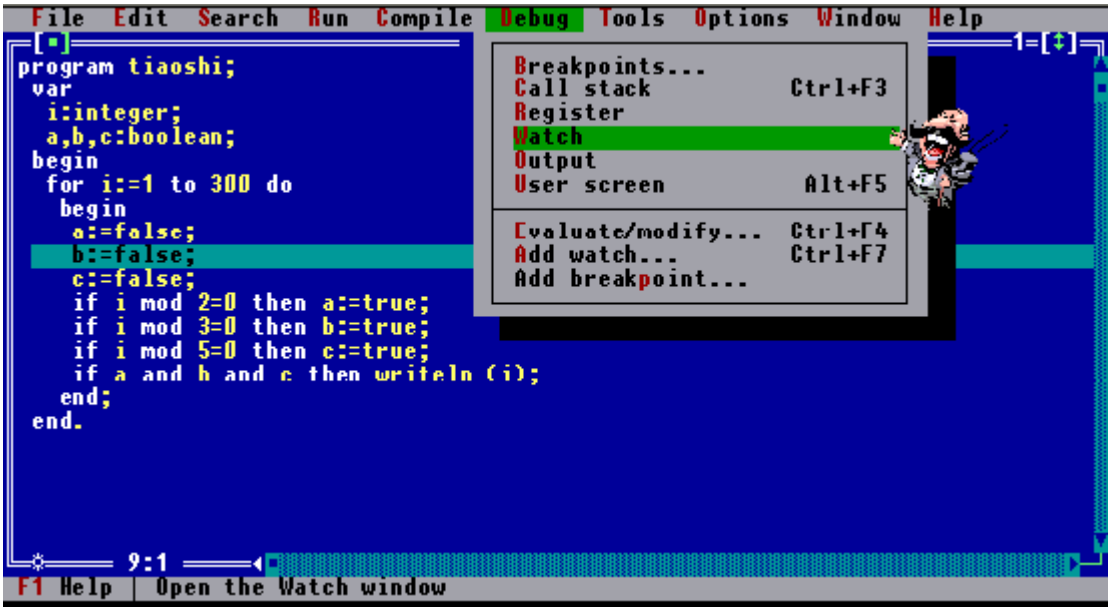
```
                    if i mod 5 = 0 then
                                writeln(i);
    end;
end.
```

该程序是输出 300 以内同时能被 2，3，5 整除的整数。 现在我们开始调试。 调试有多种方法，先介绍一种，权且叫步骤法，步骤法就是模拟计算机的运算，把程序每一步执行的情况都反映出来。通常，我们有 F8 即 STEP 这个功能来实现，如图： 不断地按 F8，计算机就会一步步地执行程序，直到执行到最后的“end.”为止。

可能你还没有发现 F8 的威力，我们不妨把上面的程序略微修改一下，再配合另外的一种调试的利器 watch，你就会发现步骤法的用处。

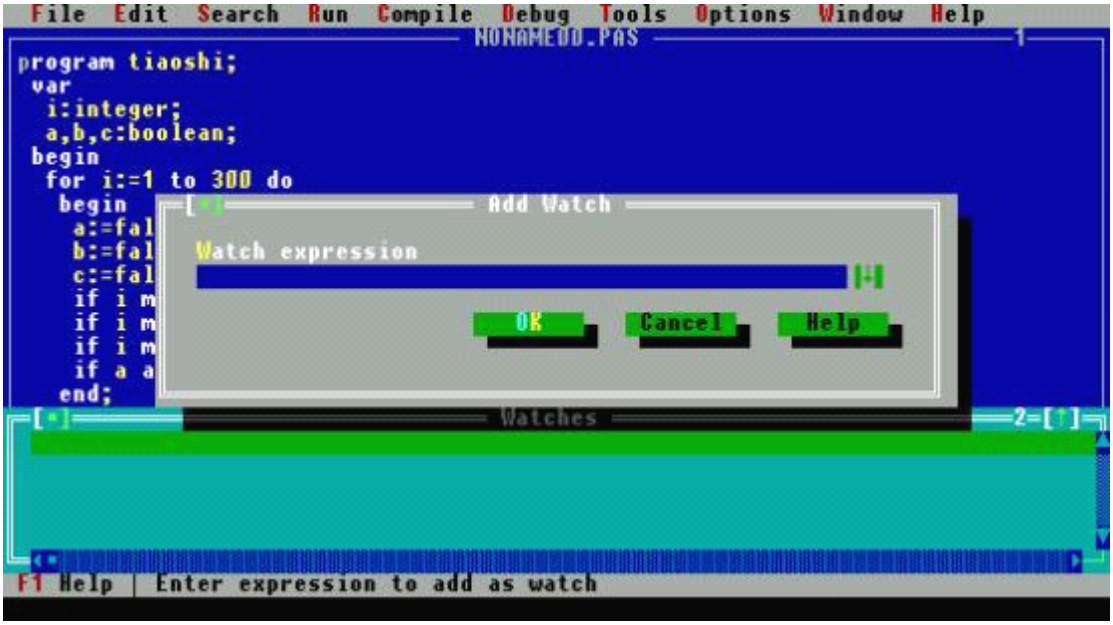
```
program tiaoshi;
var i:integer;
    a,b,c:boolean;
begin
     for i:=1 to 300 do
     begin
           a:=false;
           b:=false;
           c:=false;
           if i mod 2 = 0 then a:=true;
           if i mod 3 = 0 then b:=true;
           if i mod 5 = 0 then c:=true;
           if a and b and c then writeln(i);
     end;
end.
```

如图，我们单击菜单栏中 debug 选项，里面有一项叫 watch 的选项，我们单击它。

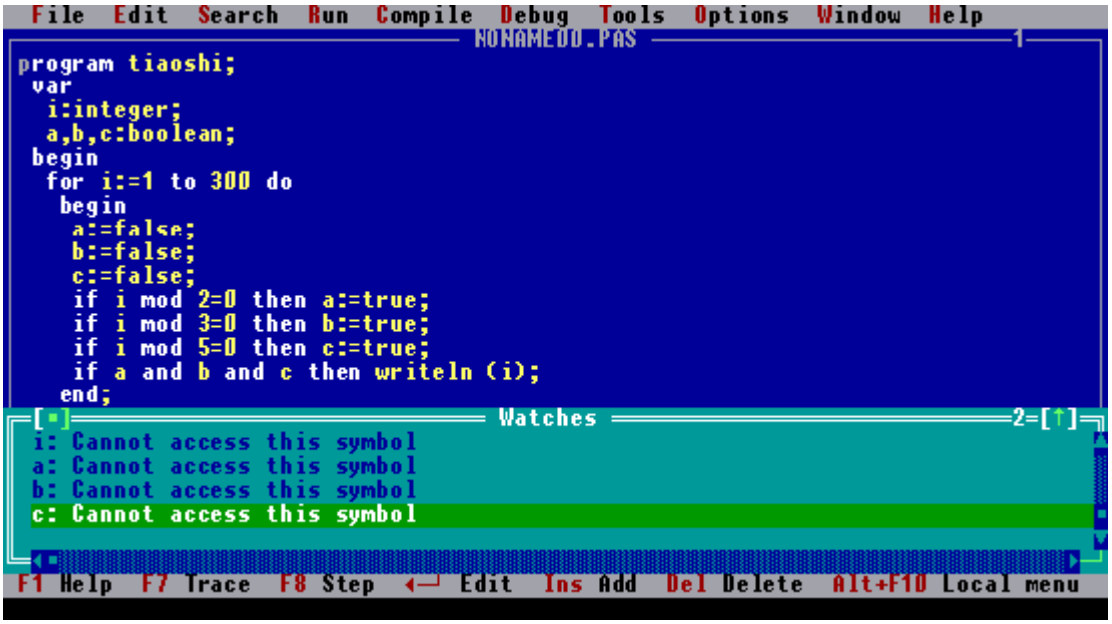


就会出现一个 watch 窗口：

watch 窗口可以让我们观察变量的变化情况，具体操作是在 watches 窗口内按 Insert 键：



这时，屏幕上弹出一个菜单，我们输入所需要观察的变量名，我们分别输入 i,a,b,c 这 4 个变量名，于是 watches 窗口内就有如下的 4 个变量的状态：



这时，我们再次使用步骤法，我们会发现，这 4 个变量的状态随着

程序的执行而不断变化，比如：

这样我们就可以方便地知道执行每一步之后，程序的各个变量的变化情况，从中我们可以知道我们的程序是否出错，在哪里出错，方便我们及时地修改。

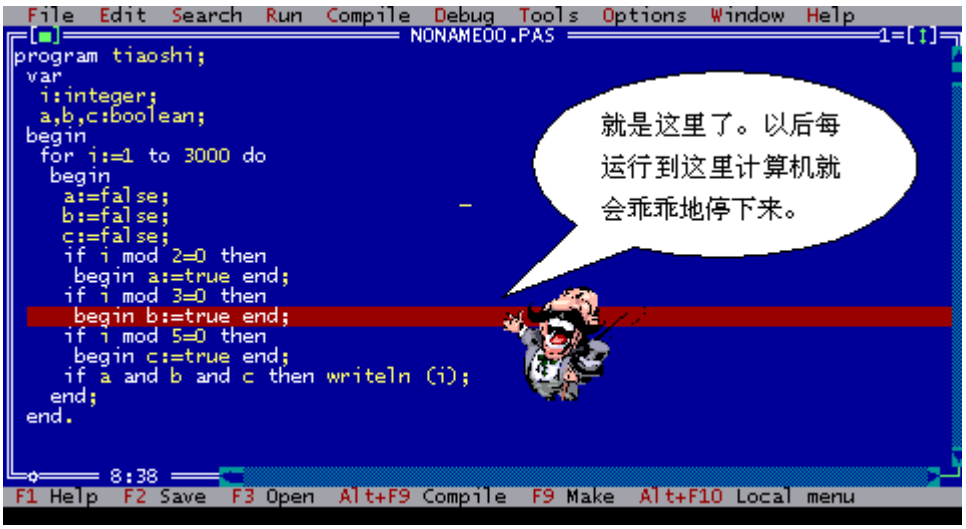
三、程序的调试(断点调试)

在前面我们已经学习了基本的程序调试方法——步骤法。步骤法有一个缺点，就是在遇到循环次数比较多或者语句比较多的时候，用起来比较费时，今天我们来学习一种新的也是常用的调试方法——断点法。

所谓断点法，就是在程序执行到某一行的时候，计算机自动停止运行，并保留这时各变量的状态，方便我们检查，校对。我们还是以前面求

同时能被 2，3，5 整除的 3000 以内的自然数为例，具体操作如下：

我们把光标移动到程序的第 14 行，按下 ctrl+F8，这时我们会发现，该行变成红色，这表明该行已经被设置成断点行，当我们每次运行到第 14 行的时候，计算机都会自动停下来供我们调试。



我们必须学以致用，赶快运用刚学的 watch 方法，看看这家伙到底有多厉害。

(二)

请记住，计算机是执行到断点行之前的一行，断点行并没有执行，所以这时 b:=true 这一句并没有执行。

断点行除了有以上用处之外，还有另外一个重要用处。它方便我们判断某个语句有没有执行或者是不是在正确的时刻执行，因为有时程序由于人为的疏忽，可能在循环或者递归时出现我们无法预料的混乱，这时候通过断点法，我们就能够判断程序是不是依照我们预期的顺序执行。

第四讲:程序设计的方法

程序设计的方法

1.模块化：

(1) 把一个较大的程序划分为若干子程序，每一个子程序解决一个总是独立成为一个模块；

(2) 每一个模块又可继续划分为更小的子模块；

(3) 程序具有一种层次结构。

注：运用这种编程方法，考虑问题必须先进行整体分析，避免边写边想。

2.自顶向下：

(1) 先设计第一层(即：顶层)，然后步步深入，逐层细分，逐步求精，直到整个问题可用程序设计语言明确地描述出来为止。

(2) 步骤： 首先对问题进行仔细分析，确定其输入、输出数据，写出程序运行的主要过程和任务； 然后从大的功能方面把一个问题的解决过程分成几个问题，每个子问题形成一个模块。

(3) 特点：先整体后局部，先抽象后具体。

3.自底向上：

(1) 即先设计底层，最后设计顶层；

(2) 优点：由表及里、由浅入深地解决问题；

(3) 不足：在逐步细化的过程中可能发现原来的分解细化不够完善；

(4) 注意：该方法主要用于修改、优化或扩充一个程序。

4.例子：求 1 到 n 之间的素数。

解：要求 1 到 n 之间的素数，程序要做的事就是从 1 开始依次找，判断是否是素数，是则打印出来，否则继续往下找，直到 n 为止。于是初步设想成：

```
begin
  read(n);
  number:=2;
  while number〈n do
  begin
    if number 是一个素数 then write(number);
      number 取下一个值；
    end
 end.第二步：细化“number 是一个素数”及“number 取下一个值”。
```

(1) 细化“number 是一个素数”：

“number 是一个素数”这是一个布尔值，当 number 是一个素数时为

true，否则为 false。细化如下：

```
k:=2;
lim:=number-1;
repeat
  if nubmer 能被 k 整除 then
    prim:=false
  else begin
    k:=k+1;
    prim:=true;
  end;
until not(prim) or (k 达到 lim)；
```

(2) 细化“number 取下一个值”：

```
number:=number+1;
```

第三步：细化“number 能被 k 整除”及“k 达到 lim”。

(1) 细化“number 能被 k 整除”：

```
number mod k=0;
```

(2) 细化“k 达到 lim”：

```
k<=lim;
```

第四步：补充完整程序。

第五步：从所有的素数除了 2 之外都是奇数的角度出发优化程序。

程序设计步骤:

1.分析问题：

对要解决的问题，首先必须分析清楚，明确题目的要求，列出所有已知量，找出题目的求解范围、解的精度等。例“第 10 周练习”第 7 题——兔子的繁殖问题，必须找出其繁殖规律。

2.建立数学模型：

对实际问题进行分析之后，找出它的内在规律，就可以建立数学模型。只有建立了模型的问题，才能可能利用计算机来解决。如上例，可推出递推公式 u[n]=u[n-1]+u[n-2](这是菲波那契数列)

3.选择算法：

建立数学模型后，还不能着手编程序，必须根据数据结构，解决问题的算法。一般选择算法要注意：

(1) 算法的逻辑结构尽可能简单；

(2) 算法所要求的存贮量应尽可能少；

(3) 避免不必要的循环，减少算法的执行时间；

(4) 在满足题目条件要求下，使所需的计算量最小。

4.编写程序： 把整个程序看作一个整体，先全局后局部，自顶向下，一层一层分解处理，如果某些子问题的算法相同而仅参数不同，可以用子程序来表示。

5.调试运行；

6.分析结果；

7.写出程序的文档：

主要是对程序中的变量、函数或过程作必要的说明，解释编程思路，画出框图，讨论运行结果等。

8.例 1：输入奇数 n，计算并输出 n 位的魔方阵。

说明：

(1) 魔方阵就是 n*n 个不同的正整数按方阵排列时，它的每一行，每一列以及沿对角线的几个数的和具有同一性质的方阵。

(2) 由 1 到 n*n 个自然数数构成的魔方阵是最基本的，又称为“幻方”，这种方阵的每行、每列和每个对角线上的元素的和全部相等，亦即等于一个常数。该常数是 n(n*n+1)/2。

(3) 方法： 首先确定 1 的位置，通常放在第一行的中间位置； 然后当前自然数的右上方放下一个自然数； 如果当前自然数在第一行但不在最右侧，则下一个自然数在最后一行，列数右移一列； 如果当前自然数在第一行最右侧，则下一个自然数在当前自然数的下侧； 如果当前自然数在其它行的最右侧，则下一个自然数在上一行的最左侧。

9.例 2：任何一个整数的立方都可以写成一串奇数之和。

说明：

(1)这是著名的尼科梅切斯定理。即 1^3=1 2^3=3+5=8 3^3=7+9+11=27

……

(2)数据间关系的规律：

·n^3 是 n 个奇数之和，如 2^3 是 2 个奇数之和，3^3 是 3 个奇数之和；

·这 n 个奇数是相邻的，只要知道各式的第一个奇数也就知道所有的 n 个奇数：

组成 1^3 的 1 个奇数是奇数序列中的第 1 个奇数；

组成 2^3 的 2 个奇数中最大的奇数是奇数序列中的第 3(3=1+2)个奇数(值为 5)；

组成 3^3 的 3 个奇数中最大的奇数是奇数序列中的第 6(6=1+2+3)个奇数(值为 11)；

由此推出：

组成 n^3 的 n 个奇数中最大的奇数是奇数序列中的第 m(m=1+2+3+...+n)个奇数，即： m=n(n+1)/2

·奇数序列中第 m 个奇数的值是：modd=2m-1 modd 表示“第 m 个奇数”，是组成 n^3 的奇数序列中最大的一个奇数。

例如，第 2 个奇数是 3，第 6 个奇数是 11。

·n^3=modd+(modd-2)+(modd-4)+...(modd-2(n-1))

第五讲:赋值语句

赋值语句

〖语法分析〗

PASCAL 有两个语句可以改变变量的值。赋值语句是其中之一（另一个是读语句）。赋值，顾名思义，就是把一个值赋予某个量。可以这理解：变量相当于装东西的容器，赋值的过程就是把东西放进容器的过程。赋值语句格式如下：

变量：=表达式；

写赋值语句有以下几点要注意：

1、赋值号“：=”

赋值号由两个字符构成，是一个运算符。如果把这两个字符拆开，那么这两个字符就是别的意思了：“：”是分隔符而“=”是关系运算符，判定两个对象是否相等。刚刚写程序的同学要特别注意这一点。

例：a，b：integer；——是一个说明语句。“：”是变量表和变量类型的分隔符

a=b——是一个表达式。它的值是一个布尔类型的量：TRUE 或 FALSE

a：=3；——是一个语句。把整型常量值 3 赋给整型变量 a

X：=1； 把 1 这个数值赋给 X 这个变量；

A：=B*C+12*INT（D）； 把 B*C+12*INT（D）这个表达式的值赋给变量 A；

Y：=Y+1； 把 Y+1 的值再赋给 Y 这个变量， 即累加。

以下赋值语句都是错误的：

S：=T：=1； 不能有两个赋值号在一个语句中；

X+2：=4； 赋值号左边不能是表达式；

Z=3； 少了冒号；

由上可以看出，赋值语句实际上是先把等于号右边的表达式计算出，再赋给赋值号左边的变量。

2、变量要先说明

在赋值号左边出现的变量，要在程序头的说明部先加以说明，否则编译时出错。

3、表达式必须要有确定的值

赋值号右边出现的表达式，必须是可以求值的。也就是说，经过运算之后，能得出一个具体的、确定的值出来。大家想一想，如果连表达式自己都不知道自己的值是多少，怎么还能把值“赋予”别人呢？

以下程序的作用是先把 A，B 两个变量赋上值（3 和 2），打印在屏幕上；然后再交换它们的值，再打印在屏幕上：

```
Program L31;
var a,b,c:integer;
begin
a:=3;
b:=2;
writeln('a=',a,' ','b=',b);
c:=a;
a:=b;
b:=c;
writeln('a=',a,' ','b=',b);
readln;
```

end. 这边程序说明、解释，大家不用输入。

这时刚说明 A，B，C 三个变量，值均为 0

程序开始

这时 A 的值为 3

这时 B 的值为 2

此句为打印语句，结果在输出屏幕上

这时 C 的值为 3

这时 A 的值为 2

这时 B 的值为 3， 此时 A，B 的值已经交换

打印语句

等待用户敲回车键继续

程序结束

上述程序是一个比较简单的顺序结构的程序，希望大家看懂程序每一句的作用，进一步理解 PASCAL 程序的语法规则。在程序运行过程中各变量数值变化的情况如程序右边。从上述程序中还可以看出，一个变量如果本身已经被赋过值，当再次赋值时，原来的值就不起作用，而被新的值所覆盖。

4、赋值号两边的数据类型必须相同或相容

我们知道，PASCAL 中的量不管是变量还是常量都有一个属性称为“数据类型”。数据类型相同的或相容的才可以相互赋值。

怎么来理解这句话呢？打个比方，我们沏功夫茶用的是小茶杯，装饭时用饭碗。如果用饭碗来泡功夫茶，用小茶杯来装饭，那情形一定很滑稽而且是不可行的。回到 PASCAL 中来，赋值号左边变量如果是整型，右边表达式的值的类型也要是整型；赋值号左边变量如果是字符型，右边表达式的值的类型也要是字符型……否则的话，也要出错了。这是数据类型相同的情况。

对于数据类型相容的，我们也可以用一个例子来帮助理解。我们都喝过功夫茶，也喝过大杯茶。把功夫茶倒在大茶杯里，一般不会出什么问题；但如果把大杯里的茶倒在功夫茶杯里呢？可能小茶杯装不下大茶杯里的茶，茶“溢出”了。在 PASCAL 中也会出现这种情况。当一种数据类型的取值范围包含着另一种数据类型的取值范围时，就可能出现类型相容的情况。如实型与整型，整型、字符型与它们各自的子界类型……如果把整型值赋给实型变量，把整型子界值赋给整型变量，不会出错；但如果反过来，就会出现“溢出”，出错了。

因些，我们在写赋值语句时，要注意两边的类型是否匹配。

例：有程序如下：

```
Program q32;

Var I,j,k:integer;

R:real;

S:string;

Begin

I:=123.456;
```

R:=146;

R:=trunc(1231.54346);

K:=6/3;

S:=123;

S:= ’ aasd ’ +1234;

End.

说明 I， J， K 三个变量是整数型；

R 是实数型；

S 是字符串型；

程序开始

错误：因为 I 是整数型变量，不能把一个实数型数值赋给它；

正确： 146 也是一个实数；

正确： trunc 这个函数的返回值是整数型的，可以赋给实数型变量；

错误： 6/3 虽然得到的值是 2，但这个 2 是实数型，而不整数型的；

错误： S 是字符串型变量，而 123 是数值型值；

错误：表达式中不能把字符串与数值相加。

程序结束

总结：

与数学上的“=”不同。X：=X+1，指将 X 的值加上 1 后再赋给 X。

PASCAL 中的“=”为关系运算符。X=(X+1)为假。

赋值号两边的数据类型必须相同。

变量必须赋初值后才能引用。(系统默认数字为 0，字符为空格，布尔值为 FALSE)

例 1：下列赋值语句那些是错的？错在哪里？（var : x,y,z,a: integer; ）

① x:=x+y　　② x=y+z　　③ x+y:=z

例 2：写出下列程序的运行结果

```
var a,b,c:integer;

begin

a:=5; b:=8;

c:=a; a:=b; b:=c;

writeln('a=',a,'? ','b=',b)

end.
```

思考题：如果不引入第三个中间变量，能否交换两个变量的值？编程序试一试。

学习两个过程：

① inc(x,n)　等价于 x:=x+n　　② dec(x,n)　等价于 x:=x-n

inc(x)　等价于 x:=x+1　　dec(x)　等价于 x:=x-1?

第六讲:输入（读）read 语句

输入（读）read 语句

〖语法分析〗

读语句（read 语句）和赋值语句一样，能够改变变量的值。与赋值语句不同，读语句从键盘或文件接收值赋予变量，而赋值语句则直接由程序语句获得。读语句格式如下：

read(变量名表)；

readln(变量名表)；

readln；

读语句是编程中用得最多的语句之一。在使用时有几点要注意：

1、变量名表。写在括号中的变量，都要在变量说明中先预以说明；变量与变量之间，以“,”分隔；

例：

```
var a,b:integer;
read(a,b);
```

2、从键盘接收数据时，要注意各种不同数据类型数据的分隔符不同。所谓分隔符就是两个完整的数值之间的标记，也可以这样理解，当计算机从键盘读入数据时，一旦碰到分隔符，就认为当前的数据读入已完成，可以把它赋给相应的变量了。各种数据类型的分隔符如下：

数值型（包括整型、实型以及它们的子界类型）以空格或回车符作为分隔符；

字符型不需分隔符（因为字符型数据的长度固定，只有一个）；

字符串以回车符作为分隔符。

3、注意 read 与 readln 的区别

假设 READ 或 READLN 要求输入 X 个数据，而我们输入时也输入了 X 个数据，这时 READ 与 READLN 的功能是完全一样的。

但是，如果要求输入 X 个数据，而我们输入了多于 X 个数据时（如 X+N 个），这时 READ 与 READLN 都是把前 X 个数据赋给 X 个变量，而剩下的多输入的 N 个数据就有如下情况：如果是 READ，则这 N 个数据被保存下来，到下一个 READ/READLN 语句时再赋给剩下的变量；而用的是 READLN 的话，这些数据将被放弃，不会留至下一个 READ/READLN。

例：有两段程序有相同的变量说明如下，不同的读语句，我们可以通过比较它们执行结果的异同来理解 read 与 readln 的区别。

变量说明 var a,b,c,d:integer;　　执行结果

a b c d

输入数据 1 2 3 4 5

6 7 8

程序段一 read(a);

readln(b,c);

read(d);

1 2 3 6

程序段二 readln(a);

read(b,c);

read(d) 1 6 7 8

在程序段一执行时，“read(a);”语句接收了第一个数据 1 并将它赋给变量 a；接着执行第二个语句“readln(b,c);”，接收了第一行数据中的 2、3 并把它们分别赋给变量 b,c，同时，把本行其它数据全部屏蔽掉，也就是宣布它们全部作废。程序段二的执行情况也是如此。

因此,我们可以得出结论:语句 read 只管接收数据,语句 readln 接收完数据后，还把同行的其它数据全部宣布作废。

4、“readln;”语句从键盘接收一个回车符。这个语句通常用在需要暂停的地方。如输出时用来等待程序员看清结果。

总结：

变量类型与输入的数据类型必须一致。

输入顺序与变量顺序要一致。

注意数据个数与变量个数的关系。

第七讲:输出（写）write 语句

〖语法分析〗

写（write）语句是 Pascal 中唯一能将运算结果送出显示在显示器屏幕的语句。格式如下：

write(输出量 1，输出量 2，……)；{输出后不换行}

writeln(输出量 1，输出量 2，……)；{输出后换行}

writeln；{输出一个回车符}

使用写语句时也有一些小问题需要注意。

1、输出量可以是：

变量。输出变量的值。输出多个变量时，变量间用“,”分隔。

表达式。输出的是表达式的值。

常量。直接输出常量值。

例 1：以下两个程序段的输出分别为：

① write(1,2,3,4);write(5,6);　　② writeln(1,2,3,4);write(5,6);

输出为：　　输出为：

123456　1234

例 2：有三个小朋友甲乙丙。甲有 50 粒糖果，乙有 43 粒糖果，两有 13 粒糖果。现在他们做一个游戏。从甲开始，将自己的糖分三份，自己留一份，其余两份分别给乙与丙，多余的糖果自己吃掉，然后乙与丙也依次这样做。问最后甲、乙、丙三人各有多少粒糖果？

分析：在游戏中每个小朋友拥有的的糖果数是在变化的，用 a,b,c 三个变量分别存放甲乙丙在某一时刻所拥有的糖果数。对于每人，分糖后，他的糖果数一定为原来的糖果数 div 3（用整除恰可以表示多余的糖自己吃掉）。而其他两人则增加与这个小朋友现在拥有的糖果数。

程序如下：

```
var A,B,C:integer;
begin
   A:=50; B:=43; C:=13;                    {初始糖果数}
   A:=A div 3; B:=B+A; C:=C+A;             {甲分糖果后，每个人拥有的糖果数变化情况}
   B:=B div 3; A:=A+B; C:=C+B;             {乙分糖果后，每个人拥有的糖果数变化情况}
   C:=C div 3; A:=A+C; B:=B+C;             {丙分糖果后，每个人拥有的糖果数变化情况}
```

```
    writeln('A=',A,'    B=',B,'    C=',C);
end.
```

2、场宽的限制在输出不同格式的数值时的作用：

例 3：输出多个空格。

write(''：n);句子的意思是以 n 个字符宽度输出冒号前数据项，如果数据项长度不足 n，则前面以空格补齐；如果数据项长度大于 n，则以实际长度输出。如上语句句输出 n 个空格。

例 4：数据项间隔。

如输出最多四位的数据：write(x:5)。则数据间至少分隔一个空格。

例 5：实型数据小数位数的确定。

实型数据不带格式限制时，以科学计数法的形式输出，和我们的一般书写习惯不同。如果加上场宽的限制，则可以有不同的效果：

```
var a:real;
begin
    a:=15/8;
    writeln(a);{输出 1.8750000000E+00}
    wiiteln(a:0:2);{输出 1.88 整数部分按实际位数输出，小数部分保留两位小数，末位四舍五入.}
    writeln(a:0:0):{输出 2 只输出整数部分，小数部分四舍五入}
end.
```

例 6、写出下列程序的运行结果：

```
const s=’abcd’;

var i:integer;     r:real; c:char; b:Boolean;

begin

        i:=1234; r:=123.456;     c:= ’*’; b:=true;

        writeln(i,i:5,i:3);

        writeln(r,r:8:4,i:8:2);

        writeln(c,c:4);

        writeln(s,s:8,s:3);

        writeln(b,b:5,b:3)
```

end.

运行结果

1234 12341234

1.2345600000E+02123.4560 123.46

* *

abcd abcdabcd

TRUE TRUETRUE

3、“writeln;”语句通常用于输出多组数据时在屏幕上输出空行来分隔数据组。

总结：

输出数据也根据定义的场宽向右靠齐。

当组成输出数据的字符个数小于场宽时，在数据值的左边补足空格。

当组成输出数据的字符个数大于场宽时，系统将自动向右边伸至恰好的大小位置，突破原来定义的场宽大小。

相关文章:

pascal 第六讲:输入（读）read 语句

PASCA 第五讲:赋值语句

pascal 第四讲:程序设计的方法

Pascal 第三讲:程序的调试

PASCAL 第二讲:程序的基本结构

相关评论

流云于 2005-5-2 16:30:21

练习题

1、下面三段程序运行结果一样吗？

Var a:integer;

```
B:real;

Begin

A:=2;

B:=2;

Writeln(b/a);

End.
 Var a:integer;

B:real;

Begin

A:=2;

B:=2;
```

```
Writeln(a/a);


End.
 Var a:integer;


B:real;


Begin


A:=2;


B:=2;


Writeln(a div a);


End.
```

2、如果 a,b,c 的值分别是 2， 4， 6。执行下列语句后， a,b,c 的值各是多少？

```
A:=b;b:=c;c:=a;
```

B:=c;c:=a;a:=b;

C:=a;a:=b;b:=c;

T:=a;a:=b;b:=c;c:=t;

3、请说明下列两个变量的值有何不同：

C ： =‘’ {单引号间没有字符 }

C ： =‘ ’ {单引号间有一个空格 }

4、写出下列每小段程序的运行结果（只是程序的一部分）：

A.write(‘ * ’);write(‘ * ’);write(‘ * ’);write(‘ * ’);

B.writeln(‘ * ’);writeln(‘ * ’);writeln(‘ * ’);writeln(‘ * ’);

C.write(‘ * ’);writeln;write(‘ * ’);writeln;write(‘ * ’);writeln;write(‘ * ’);writeln;

D.write(‘’);writeln(‘ * ’);write(‘’);writeln(‘ * ’);write(‘’);writeln(‘ * ’);write(‘’);writeln(‘ * ’);

5、编一程序，从键盘输入整数 A， B 的值，然后把 A， B 的值交换后输出。

6、编一程序，键盘输入整数 A， B 的值，然后打印 A 除以 B 的商的整数部分及余数。

7、编一程序，从键盘输入一个实数，然后输出其四舍五入精确到小数点后第 3 位的数。

8、编一程序，从键盘输入一个四位整数，然后把该数的每一位进行交换，即千位换为百个位，百位换成十位，然后输出新数。

9、 random 函数可得到 0 到 1 之间的随机小数，请编一程序用此函数得到 10 到 100 之间的随机整数。
（注意： randomize 可初始化随机数）。

10、从键盘输入一个多位整数，要求打印出它的各位数之和。

11、从键盘输入两个整数，打印出更小的那个数。

12、从键盘输入长方形的两个边长 a 和 b，计算它的周长并输出到屏幕上；（整型，输入语句，赋值语句，输出语句）

13、输入一个时、分、秒，把它转换为一个秒数输出；（表达式）

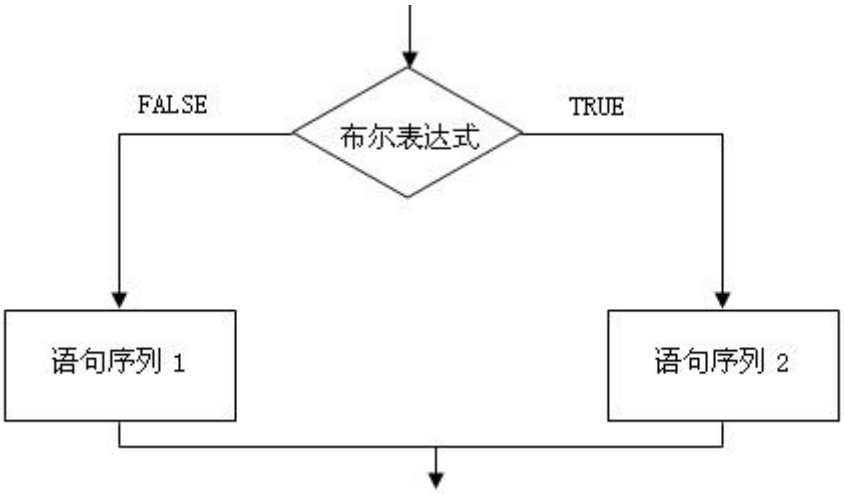
14、输入一个二位整数，将它的各位数字倒序输出；（MOD，DIV）

15、从键盘输入圆的半径 r，计算它的周长和面积并输出到屏幕上；（实型，输出语句，实型的输出格式）

第八讲:条件（IF）语句

条件 (IF) 语句

条件语句是用一个布尔表达式的值来决定程序的走向。即程序提供一个分支，由布尔表达式的值来决定程序究竟运行哪个分支。即程序已经不是顺序结构了，而是提供了一个分支结构。每次只可能运行其中的一个分支。如下图所示：



条件语句有两种格式：

1、 if <布尔表达式 > then <语句 >；

2、 if <布尔表达式 > then <语句 1> else <语句 2>；

格式 1 的作用是：如果布尔表达式值为 TRUE（即该条件满足），则运行语句，否则不运行任何语句。

格式 2 的作用是：如果布尔表达式值为 TRUE（即该条件满足），由运行语句 1，否则（即布尔值为 FALSE，亦即条件不满足）就运行语句 2。注意：只有条件语句结束时才有分号。

注意：如果条件语句中的分支语句不止一句，可用“ begin …… end;”

来框住，加以区别。如：以下两个程序都为从键盘输入 1 个整数，打印出它的算术平方根。 [例 4、 2]：

```
Program q421;
Var a:integer;
Begin
   Readln(a);
   If (a<0) then
   Begin
      Writeln( ‘ WRONG! ’ );
   End else
   Begin
      Writeln(a);        Writeln(sqrt(a):8:2);
   End;
End.  Program q421;
Var a:integer;
Begin
Readln(a);
If (a<0) then Writeln( ‘ WRONG! ’ ) else Writeln(sqrt(a):8:2);
End.
```

条件语句的嵌套：条件语句是可以嵌套的，例如：

```
if a>0 then if a>1 then …… else …… else … .;
```

上述语句看起来无法理解，但如果我们把它改写为以下形式时，就一目了然了：

```
if a>0 then begin
      if a>1 then begin
         ……
      end
else begin
         ……
      end
else begin
      ……
end;
```

也就是说，第二个 IF 语句只是第一个 IF 语句中的一部分，即嵌入的一个 IF 语句。

例：输入一个年份，判断它是否闰年。我们知道，每四年中有一年是闰年，即有 366 天，而其余三年是平年，只有 365 天，而按照规定：

1、如果哪一年的年份能被 4 整除，则该年一般为闰年； 2、这样，每 100 年又会少一天，所以又规定如果该年能被 4 整除，又能被 100 整除，则认为该年不是闰年，而是平年； 3、这样，每 400 年又会多出一天，所以又规定，如果哪一年能被 400 整除，则该年又是闰年。

由上可知：1980，1996，1984 年均为闰年，1900 年为平年，而 2000 年为闰年。

程序如下： [例 4、 3]

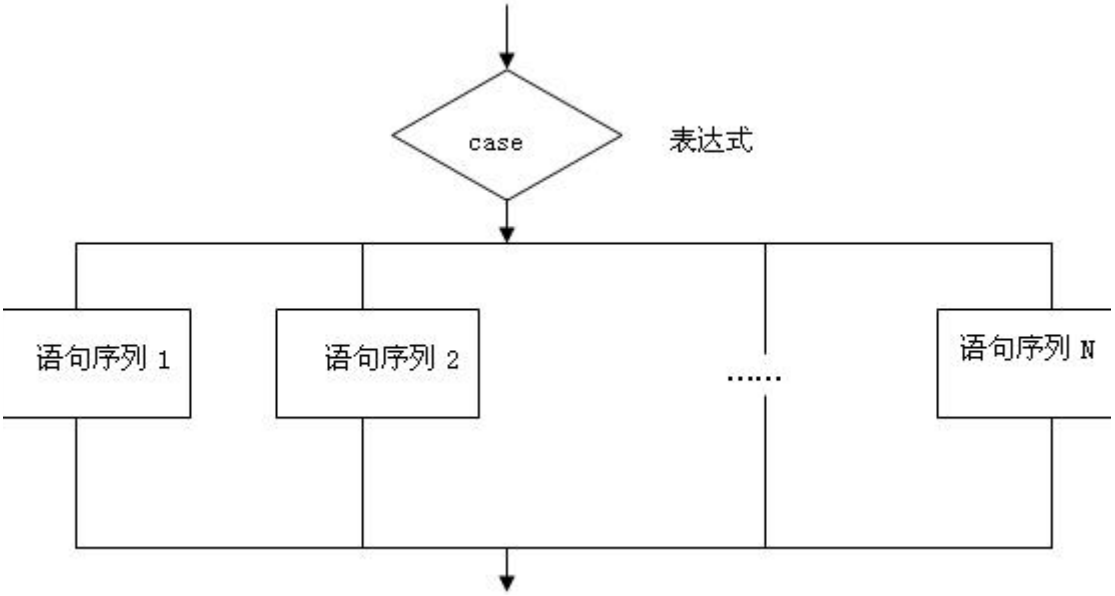
```
Program q43;
Var y:integer;
Begin
    Readln(y);
    if (y mod 4=0) then begin
      if (y mod 100=0) then begin
        if (y mod 400=0) then begin
          writeln('run');
        end else begin
          writeln('ping');
        end;
      end else begin
```

```
            writeln('run');
          end;
        end else begin
           writeln('ping');
        end;
    end.
```

第九讲:CASE 多分支语句

多分支语句

IF 语句只能使程序有两个分支，当遇到需要有两个以上分支时，IF 语句就很不适用了，这时我们可以用 CASE 语句，它能使程序有很多个分支。其效果如下图：



CASE 表达式 OF

常量 1：语句序列 1；

常量 2：语句序列 2；

……

常量 N：语句序列 N；

else 语句序列 N+1；

END；

其中语句序列如果只有一句，可直接写在常量后的冒号后；如果语句序列有很多句，则应该用 BEGIN…… END 加以框住。

CASE 语句的作用是，根据表达式，表达式可以有多个值，分别对应于常量 1、 2 等值时，就运行该常量后所对应的语句。

例：输入 1 到 7 之间的某个整数，打印出其对应的星期的英语名：[例 4、 4]：

```
program q44;
var n:integer;
begin
   write('n=');
   readln(n);
   case n of
      1:writeln('Monday');
      2:writeln('Tuesday');
      3:writeln('Wednseday');
      4:writeln('Thursday');
      5:writeln('Friday');
      6:writeln('Saturday');
      7:writeln('Sunday');
      else writeln('WORNG!');
   end;
end.
```

例：输入一个学生的数值化成绩，把它转化为等级化成绩。即 100-90 是 A， 89-80 是 B， 79-70 是 C， 69-60 是 D， 59-0 是 E。

```
Program q44;
Var s:integer;
  G:char;
Begin
  Write('Input the score:');
  Readln(s);
  Case s div 10 of
    10,9:g:='A';
    8:g:='B';
    7:g:='C';
    6:g:='D';
    else g:='E';
  end;
  writeln(s,'  ',g);
end.
```

用 S DIV 10 这个表达式作条件；

当 S DIV 10 等于 10 或 9 时，表示 S 是 90 至于 100 之间的值

相关评论

流云于 2005-5-2 16:45:25

练　习

1、设 X，Y，Z 的值分别是 FALSE，TRUE，FLASE。写出下列逻辑表达式的值：

not x and not y;

true and x or y;

(x and z) or (z and y);

x or z and y;

2、 编写一个程序，功能是从键盘输入一个整数，判断它是否二位数，如果是，就打印它，然后结束程序，否则继续要求输入数。

3、 编写一个程序，功能是从键盘输入三个整数，打印出其中最大的一个值。

4、从键盘读入一个数，判断它的正负。是正数，则输出"+"，是负数，则输出"-"

5、输入 a,b,c 三个不同的数，将它们按由小到大的顺序输出

6、铁路托运行李规定：行李重不超过 50 公斤的，托运费按每公斤 0.15 元计费；如超 50 公斤，超过部分每公斤加收 0.10 元。编一程序完成自动计费工作。

7、打印某年某月有多少天。（提示：A、闰年的计算方法：年数能被 4 整除，并且不能被 100 整除；或者能被 400 整除的整数年份。B、利用 MOD 运算可以判断一个数能否被另一个数整除）

8、某超市为了促销，规定：购物不足 50 元的按原价付款，超过 50 不足 100 的

按九折付款，超过 100 元的，超过部分按八折付款。编一程序完成超市的自动计费的工作。

9、 编写一个程序，功能是从键盘输入 1—12 中的某一个数字，由电脑打印出其对应的月份的英语名称。

10、以下程序的功能是从键盘输入一个式子，它只有三个字符，第一个及第三个都是数字，中间那个是运算符，程序能把它的结果打印出来。试在程序空中填上相应的语句。

注意：计算机是不懂得把输入的字符串进行计算的，我们必须自己动手把其中的数字转化成数值型的数据，这其中我们使用了 VAL 函数，如程序中的：Val(s[1],a,c)，是把 S 字符串中的第一个字符转换为数值 A，同时输出了一个错误代码 C（其实对我们是没有用的）。

```
Var a,b,c,d:real;
  S:string[3];
Begin
  Readln(s);
  Val(s[1],a,c);                    ;
  Case s[2] of
    '+':d:=a+b;
    '-':d:=a-b;
```

```
    '*':      ;
    '/':d:=a/b;
  end;
  writeln(s,'=',    );
  end.
```

把 S 定义成三个字符的字符串；

把 S 串中的第一个字符转换为数值；

把 S 串中的第三个字符转换为数值；

用 S 串中的第二个字符作为条件表达式；

第十讲：REPEAT 循环

直到循环（REPEAT-UNTIL）类型

直到循环语句的语法格式是：

repeat 语句序列；

（循环体）

until 布尔表达式；

其作用是：重复执行语句序列（循环体），直到布尔表达式的值为 TRUE 为止。即，当执行完一次语句序列后，布尔表达式的值已经为 TRUE 了，这时循环将不会再被执行，而转向执行 UNTIL 语句以下的语句。

例：计算 M=1+2+3+4+……，直到 M 的值大于 5050 为止。 [例： 5、1]

```
var m,I:integer;
begin
   m:=0;
   I:=0;
   Repeat
      I:=I+1;
      M:=m+I;
   Until m>5050;
   Writeln(I,’ ’,m);
End.
```

在上述程序中，我们使用了 PASCAL 中的两个作用强大的概念：累加和循环。

累加：我们在上述程序中没定了两个累加器： I， M。累加器初值我们在第一句中设定了为 0，之后每次运行一次 I： =I+1 后， I 的值就比原来大 1。每运行一次 M： =M+I 后， M 的值就被 M+I 所替代。

循环：这里我们使用的是直到循环，即重复执行循环体中的两个语句，直到 M>5050 这个条件满足为止（即这个布尔表达式的值为 TRUE 为止）。

上述程序运行后，会在输出屏幕上显示两个值： 101 5151。即当 I 的值为 101 时，这时所算得的 M 的值为 5151。亦即，1+2+3+…… 101=5151。

象上述这种 X=1+2+3+……这种加法，我们就把它叫累加，这里的 X 就是累加器，一般初值为 0。

而 N=1*2*3*4*5*6*……这种乘法我们把它叫累乘，这里 N 就是累乘器，一般初值为 1（为什么？）。

一般的：1*2*3*…… N，我们把这个式子的结果叫做 N 的阶乘（ N!）。

如： 4！ =1*2*3*4。

例：计算 1 8！ [例 5、 2]

```
var I:integer;
  x:longint;
begin
  I:=0;
  X:=1;
  Repeat
    I:=I+1;
    X:=x*I;
  Until I=18;
  Writeln(x);
End.
```

当程序开始时， I 的初值被定为 0， X 的初值被定为 1（累乘器）。然而开始进入循环，每次 I 的值比原来增加 1，然后再乘进 X 中去，直到 I 等于 20 时，最后一次把 I 乘进 X 后，这时 I=20 这个条件已经满足（ I=20 的值已经为 TRUE），所以循环就被退出，而程序转向执行 UBTIL 以后的语句： WRITELN（ X）；。

请大家想一想，为什么 X 要定义为 LONGINT 型。

练习：

1、输入一个正整数 N，把它分解成质因子相乘的形式。

如：36=1 X 2 X 2 X 3 X 3； 19=1 X 19

（提示：设因子为 I，从 2 开始到 N，让 N 重复被 I 除，如果能整除，则用商取代 N，I 为一个因子；如果不能整除，再将 I 增大，继续以上操作，直到 I 等于 N。）

第十一讲：当（WHILE）语句

当（WHILE）语句

当语句的语法格式是：

while 布尔表达式 do begin

　语句序列；（循环体）

end;

其作用是，当布尔表达式的值为 TRUE 时，才会运行语句序列（循环体），否则循环将不会被执行，即从循环头部就退出，而转向执行 END 后的语句。

例：计算 １８！ [例 5、 3]

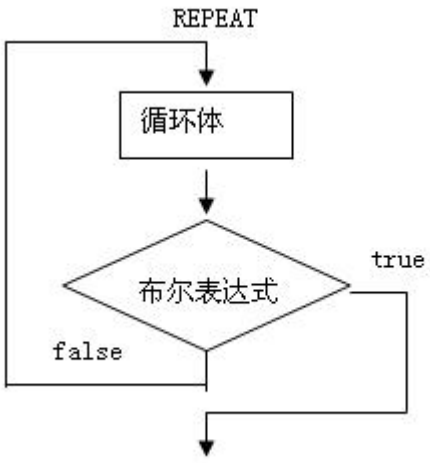
```
var I:integer;
  x:longint;
begin
  I:=0;
  X:=1;
  While I<18 do begin
    I:=I+1;
    X:=x*I;
```

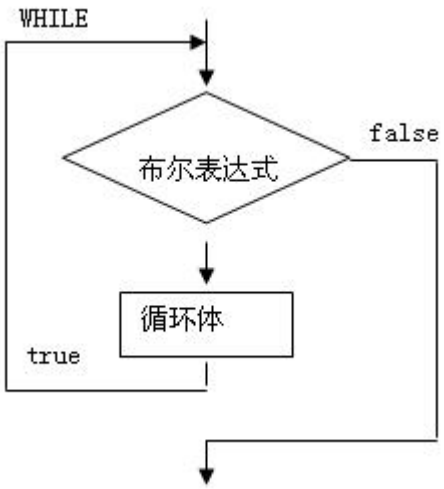
```
  End;
  Writeln(x);
End.
```

请大家把此程序与上一节的 [例 5、 2]进行比较，看两种循环在运用时有何不同。

WHILE 语句是在循环开始时就判断布尔表达式的值时否为 TRUE，如果为 TRUE，就进入循环，运行循环体，如果为 FALSE，就不运行循环体，而直接转向执行 END 后的语句 WRITELN（ X）；。

REPEAT 与 WHILE 循环的示意框图如下所示：





从上述框图中可以看出，要使用循环语句时，必须要确定循环体及条件（布尔表达式）两个重要因素，亦即首要考虑的是：我要重复执行哪些语句，我要重复到什么时候为止！

[例 5、 4]，从键盘上输入两个整数 M， N，求它们的最大公约数。

分析：我们只需从 M， N 中更小的一个数开始，每次让其减 1，直到这个数能同时被 M 和 N 数整除为止。在下述程序中，我们在程序头部调用了 CRT 单元，是为了使用 CLRSCR 语句来清屏，即把输出屏幕上的字符清除干净。

```
Uses crt;                    {调用 CRT 单元}
Var m,n,x:integer;
Begin
  Clrscr;                    {清屏}
  Write('Please input 2 numbers:')
  Readln(m,n);
  If m>n then x:=n else x:=m;
  While (n mod x<>0) or (m mod x<>0) do begin
    x:=x-1;
  End;
  Writeln(x);
```

```
End.
```

上述程序如果改用 REPEAT 语句来做的话，程序为：

```
uses crt;
var m,n,x:integer;
begin
  write('Please input 2 numbers:')
  Readln(m,n);
  If m>n then x:=n+1 else x:=m+1;
  Repeat
    X:=x-1;
  Until (m mod x=0) and (n mod x=0);
  Writeln(x);
End.
```

请大家考虑上述两个程序为何会有这样有不同之处。

练习题

1、计算下列式子的值：

（1）1+3+……+99

（2）1+2+4+8+……+128+256

2、输入一个整数，计算它各位上数字的和。（注意：是任意位的整数）

3、输入一整数 A，判断它是否质数。（提示：若从 2 到 A 的平方根的范围内，没有一个数能整除 A，则 A 是质数。）

4、求两个数的最小公倍数和最大公约数。（提示：公约数一定小于等于两数中的小数，且能整除两数中的大数。公倍数一定大于等于两数中的大数，且是大数的倍数，又能给两数中的小数整除。）

5、编写一个译码程序，把一个英语句子译成数字代码。译码规则是以数字 1 代替字母 A，数字 2 代替字母 B，……，26 代替字母 Z，如遇空格则打印一个星号‘*’，英文句子以‘.‘结束。

6、求水仙花数。所谓水仙花数，是指一个三位数 abc，如果满足 a^3+b^3+c^3=abc，则 abc 是水仙花数。

7、“百钱买百鸡”是我国古代的著名数学题。题目这样描述：3 文钱可以买 1 只公鸡，2 文钱可以买一只母鸡，1 文钱可以买 3 只小

鸡。用 100 文钱买 100 只鸡，那么各有公鸡、母鸡、小鸡多少只？与之相似，有"鸡兔同笼"问题。

第十二讲：FOR 循环语句

FOR 循环语句

前面所计的两个语句都是在未知循环次数的情况下而用的循环语句，但在程序中，如果我们已经知道循环的次数而来编程序的话，就可以使用 FOR 循环语句，这也是 PASCAL 及其它高级语言中用得最多的语句。其语法格式有两种，如下：

(1) 增量为 1：

```
for 变量名：=初值 to 终值 do begin
  语句序列（循环体）；
end;
```

(2) 增量为 -1：

```
for 变量名：=初值 downto 终值 do begin
  语句序列（循环体）；
end;
```

变量名的类型由程序头部中定义，而其初值、终值必须和它是同一类型。该变量的类型是能是有序数据类型。

上述两种格式都是用变量的初值与终值来规定循环的次数。如以下两个小程序：

Var I:integer;
Begin
For I:=1 to 5 do begin
Write(I);
End;
End.
运行结果：
12345
	Var I:integer;
	Begin
	For I:=5 downto 1 do begin
	Write(I);
	End;
	End.
	运行结果：
	54321

由上可以看出 FOR 循环的作用。

[例 5、 5]：从键盘输入一个字符串，把它按正序及逆序分别输出：

```
uses crt;
var s:string;
l,I:integer;
begin
clrscr;
write('Please input the string:');
readln(s);
l:=length(s);
```

```
for I:=1 to l do begin
   write(s[I]);
end;
writeln;
for I:=l downto 1 do begin
   write(s[I]);
end;
end.
```

[例 5、 6]计算 $1^2+2^2+.....+10^2$

```
uses crt;
var s,a:integer;
begin
clrscr;
s:=0;
for a:=1 to 10 do begin
   s:=s+a*a;
end;
writeln(s);
end.
```

[例 5、 7]：计算： 1+3+5+7+…… 101 的值：

```
uses crt;
var m,n:integer;
begin
clrscr;
m:=0;
for n:=0 to 50 do begin
   m:=m+(2*n+1);
end;
writeln(m);
end.
```

请大家注意上述程序中的几个小技巧，一个是 FOR N： =0 TO 50，共循环 51 次；一个是 M： =M+（ 2*N+1），其是的 2*N+1 得到的是一序列的奇数。

多重循环语句

多重循环语句即循环嵌套，也就是一个循环语句的循环体中还有循环语句。

[例 5、 8]编程序打印九九乘法表：

```
uses crt;
var I,j:integer;
begin
```

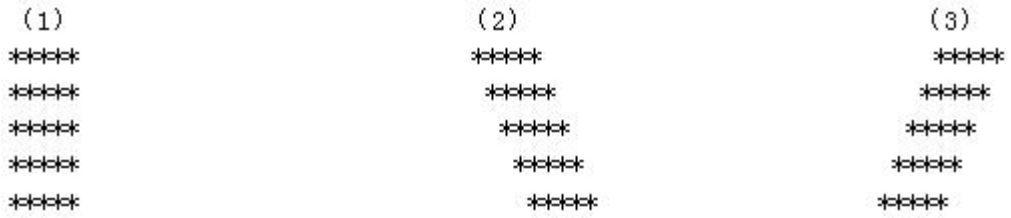
```
clrscr;
for I:=1 to 9 do begin
   for j:=1 to 9 do begin
      write(j:1,'*',i:1,'=',I*j:2);
   end;
   writeln;
end;
end.
```

运行结果：

```
1*1= 1  2*1= 2  3*1= 3  4*1= 4  5*1= 5  6*1=6   7*1= 7  8*1= 8  9*1= 9
1*2=2   2*2= 4  3*2= 6  4*2= 8  5*2=10 6*2=12 7*2=14 8*2=16 9*2=18
1*3=3   2*3= 6  3*3= 9  4*3=12 5*3=15 6*3=18 7*3=21 8*3=24 9*3=27
1*4=4   2*4= 8  3*4=12 4*4=16 5*4=20 6*4=24 7*4=28 8*4=32 9*4=36
1*5=5   2*5=10 3*5=15 4*5=20 5*5=25 6*5=30 7*5=35 8*5=40 9*5=45
1*6=6   2*6=12 3*6=18 4*6=24 5*6=30 6*6=36 7*6=42 8*6=48 9*6=54
1*7=7   2*7=14 3*7=21 4*7=28 5*7=35 6*7=42 7*7=49 8*7=56 9*7=63
1*8=8   2*8=16 3*8=24 4*8=32 5*8=40 6*8=48 7*8=56 8*8=64 9*8=72
1*9=9   2*9=18 3*9=27 4*9=36 5*9=45 6*9=54 7*9=63 8*9=72 9*9=81
```

注意：循环有嵌套时，必须分清层次，切不可把循环进行交叉。

[例 5、 9]编程序分别打印以下三图：

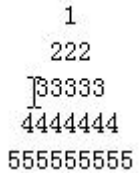


(1)	(2)	(3)
Uses crt;	Uses crt;	Uses crt;
Var I,j:integer;	Var I,j:integer;	Var I,j:integer;
Begin	Begin	Begin
Clrscr;	Clrscr;	Clrscr;
For I:=1 to 5 do begin	For I:=1 to 5 do begin	For I:=1 to 5 do begin
For j:=1 to 5 do begin	Write(‘ ’:20+i);	Write(‘ ’:20-I);
Write(‘*’);	For j:=1 to 5 do begin	For j:=1 to 5 do begin
End;	Write(‘*’);	Write(‘*’);
Writeln;	End;	End;
End;	Writeln;	Writeln;
End.	End;	End;
	End.	End.

请大家注意看清上述三个小程序，其功能是分别打印出上述三个由“ *”组成的小图形。三个图形的不同之处是：第一个直上直下，是个矩形；第二个是左斜的平行四边形；第三个是右斜的平行四边形。所以三个对应的程序也就有所不同，第（ 2）、（ 3）个程序比第（ 1）个多了一句，即显示红色的那一句。这两句的作用是使每一行开头打

印几个空格，即让图形左或右斜。 WRITELN 的作用是打印完一行后换行。

[例 5、 10]编程打印下列图形：




```
Uses crt;
Var I,j:integer;
Begin
Clrscr;
For I:=1 to 5 do begin          清屏
   Write(' ':20-I);
   For j:=1 to 2*I-1 do begin   打印每一行前的 20-I 个空格，这样每行
      Write(I:1);               都比上一行左移
   End;
   Writeln;                     每一行打印 2*I-1 列
End;                            打印该行行数
End.
```

[例 5、 11]打印 1 — 100 间的所有素数：

素数，即为除了 1 和它本身外没有另外的因数的整数。所以我们判断一个数 N 是否素数，可以用 2 至 N-1 所有数去除 N，如果没有一个数能被 N 整除，则 N 为素数。

当然，其实我们可以不用用 2 至 N-1 这第多的数去除，而只需用 2 至不大于 N 的平方根的整数去除 N 即可（为什么）？

这里我们的程序就用 2 至 TRUNC（ SQRT（ N））来判断，到后面我们还会谈到：如果要判断一个数是否素数，可只用小于 TRUNC（ SQRT（ N））的所有素数去除即可。

```
Var I,j:integer;
P:boolean;
Begin
For I:=2 to 100 do begin
   P:=true;                          循环从 2 至 100
   For j:=2 to trunc(SQRT（I）) do再判断 I 是否素数前，先把判断器
begin                                P 的值设为 TRUE
        If I mod j=0 then p:=false;
      End;                           用 2 至 trunc(SQRT（ I） )的数来
   If p then write(I:5);             除 I
End;
                                     如果 I 能整除 J，则 P 的值改为
End.                                 FALSE
```

如果 P 的值仍为 TRUE，则 I 就是素数，打印出来

[例 5、 12]以下是 A， B， C 三人说的话：

A：“ B 在说谎”； B：“ C 在说谎”； C：“ A、 B 都在说谎”。现在问，到底谁说真话，谁说谎？

分析： A、 B、 C 三人，要么说谎，要么说真话，即三个的状态要么为 FALSE，要么为 TRUE，所以，可让 A、 B、 C 三个变量进行循环，循环初值为 TRUE，终值为 FALSE。而三个人所说的话即为三个逻辑表达式，其值也为 TRUE 或 FALSE，并且与 A、 B、 C 的值是有关系的。 A、 B、 C 三个所说的话转化为逻辑表达式即为：

A： B=FLASE；

B： C=FALSE；

C：（ A=FALSE） AND （ B=FALSE）

以上三个逻辑表达式的值中， TRUE 的个数应该与 A、 B、 C 三个变量中 TRUE 的个数相同。并且，每一个变量的值应该与其对应

的话的逻辑表达式的值相等。如： A 应该等于（ B=FALSE）这个逻辑表达式。程序如下：

```
Var a,b,c:boolean;
Begin
  For a:=false to true do begin
    For b:=false to true do begin
       For c:=false to true do begin
         If (a=(b=false)) and (b=(c=false)) and
(c=((a=false) and (b=false)))
          then Begin
            Writeln(a:10,b:10,c:10);
        End;
      End;
End;
  End;
End.
```

练习：

1、计算下列式子的值：

（1）1+2+……+100

（2）1+3+5+……+97+99

2、输入一个四位数，求它各位上数字的和。

3、求水仙花数。所谓水仙花数，是指一个三位数 abc，如果满足 a^3+b^3+c^3=abc，则 abc 是水仙花数。

4、宰相的麦子：相传古印度宰相达依尔，是国际象棋的发明者。有一次，国王因为他的贡献要奖励他，问他想要什么。达依尔说：“只要在国际象棋棋盘上（共 64 格）摆上这么些麦子就行了：第一格一粒，第二格两粒，……，后面一格的麦子总是前一格麦子数的两倍，摆满整个棋盘，我就感恩不尽了。”国王一想，这还不容易，刚想答应，如果你这时在国王旁边站着，你会不会劝国王别答应，为什么？

循环结构练习题

练习

1、 随机产生一些 1—100 之间的整数，直到产生的数为 50 为止。

2、 计算 1—1000 之间能同时被 3 和 5 整除的整数的和。

3、 打印下列图形：

```
   1
  121
 12321
1234321
 12321
  121
   1
```

4、 一百匹马驮一百块瓦，一匹大马可以驮 3 块，一匹母马可驮 2 块，小马 2 匹可驮 1 块。试编程求需要各种马多少匹？

5、 有三种纪念邮票，第一种每套一张售价 2 元，第二种每套一张售价 4 元，第三种每套 9 张售价 2 元。现用 100 元买了 100 张邮票，问这三种邮票各买几张？

6、 赵、钱、孙、李、周五人围着一张圆桌吃饭。饭后，周回忆说：“吃饭时，赵坐在钱旁边，钱的左边是孙或李”；李回忆说：“钱坐在孙左边，我挨着孙坐”。结果他们一句也没有说对。请问，他们在怎样坐的？

7、找数。一个三位数，各位数字互不相同，十位数字比个位、百位数字之和还要大，且十位、百位数字之和不是质数。编程找出所有符合条件的三位数。

注：1. 不能手算后直接打印结果。

2. “质数”即“素数”，是指除 1 和自身外，再没有其它因数的大于 1 的自然数。

8、选人。一个小组共五人，分别为 A、B、C、D、E。现有一项任务，要他们中的 3 个人去完成。已知：（1）A、C 不能都去；（2）B、C 不能都不去；（3）如果 C 去了，D、E 就只能去一个，且必须去一个；（4）B、C、D 不能都去；（5）如果 B 去了，D、E 就不能都去。编程找出此项任务该由哪三人去完成的所有组合。

9、输入一个字符串，内有数字和非数字字符。如 A123X456Y7A，302ATB567BC，打印字符串中所有连续（指不含非数字字符）的数字所组成的整数，并统计共有多少个整数。

10、A、B、C 三人进入决赛，赛前 A 说：“B 和 C 得第二，我得第一”；B 说：“我进入前两名，丙得第三名”；C 说：“A 不是第二，B 不是第一”。比赛产生了一、二、三名，比赛结果显示：获得第一的选手全说对了，获得第二的选手说对了一句，获得第三的选手全说错了。编程求出 A、B、C 三名选手的名次。

11、甲、乙、丙、丁四人共有糖若干块，甲先拿出一些糖分给另外三人，使他们三人的糖数加倍；乙拿出一些糖分给另外三人，也使他们三人的糖数加倍；丙、丁也照此办理，此时甲、乙、丙、丁四人各有 16 块，编程求出四个人开始各有糖多少块。

12、截数问题: 任意一个自然数，我们可以将其平均截取成三个自然数。例如自然数 135768，可以截取成 13,57,68 三个自然数。如果某自然数不能平均截取(位数不能被 3 整除)，可将该自然数高位补零后截取。现编程从键盘上输入一个自然数 N(N 的位数<12)，计算截取后第一个数加第三个数减第二个数的结果。

13、从键盘输入一段英文，将其中的英文单词分离出来：已知单词之间的分隔符包括空格、 问号、句号(小数点)和分号。

例如：输入：There are apples; oranges and peaches on the table.

输出：there

are

apples

oranges

and

peaches

on

the

table

14、山乡希望小学收到一箱捐赠图书，邮件上署名是“兴华中学高二班”，山乡希望小学校 长送来了感谢信，可是兴华中学高二年级有四个班，校长找来了四个班的班长，问他们是哪 个班做的这件好事。一班的班长说：“是四班做的。”二班的班长说：“是三班做的好事。”三 班的班长说：“不是我们班。” 四班的班长说：“三班的班长说的不对。”

四个班的班长都说不是自己班做的，这就难坏了校长，后来得知四个班的班长中有两个 说得是真话，有两个没有说真话，请你利用计算机的逻辑判断编一个程序，找出究竟是哪个 班做了这件好事。不能手算后直接打印结果。

15、A，B，C，D，E 五个人合伙夜间捕鱼，凌晨时都疲惫不堪，各自在河边的树丛中找地 方睡着了，日上三竿，E 第一个醒来，他将鱼数了数，平分成五分，把多余的一条扔进河中， 拿走一份回家去了，D 第二个醒来，他并不知道有人已经走了,照样将鱼平分成五分，又扔 掉多余的一条，拿走自己的一份，接着 C，B，A 依次醒来，也都按同样的办法分鱼(平分成 五份，扔掉多余的一条，拿走自己的一

份)，问五人至少合伙捕到多少条鱼。

也许你能用数学办法推出鱼的条数，但我们的要求你编出一个程序，让计算机帮你算出鱼的总数。

16、试编程找出能被各位数字之和整除的一切两位数。

17、一个正整数的个位数字是 6，如果把个位数字移到首位,所得到的数是原数的 4 倍，试编程找出满足条件的最小正整数。

18、某本书的页码从 1 开始，小明算了算，总共出现了 202 个数 1，试编程求这本书一共有多少页？

19、从键盘上输入两个不超过 32767 的整数，试编程序用竖式加法形式显示计算结果。

例如: 输入 123, 85

显示: 123

+ 85

208

20、有 30 个男人女人和小孩同在一家饭馆进餐，共花了五十先令，其中男宾 3 先令，女宾 2 先令，小孩 1 先令。试编程求出男人女人小孩各多少人？

第十三讲:一维数组

一维数组

〖语法分析〗

在编程时用到一批类型相同的数据，为了处理上的方便，通常以数组的形式来定义这一批数据。

1、数组的定义格式：

```
var
    a:array [1..10] of integer;
```

其中：a 是这一批数据的名称，称为数组名；array、of 是定义数组的保留字；中括号中的数字是数据编号的下限和上限，同时也说明了数据的个数（上限-下限）；最后一个是数据的基类型，如 integer，char，real，boolean。

2、数组元素的输入：

数组名代表的并不是一个变量，而是一批变量，因而，不能直接整个数组读入，而是要逐个数组元素读入，通常用循环结构来完成这一功能。下面是几个常用输入数组元素的例子：

```
for i:=1 to 10 do read(a[i]);
    {————————从键盘读入数组元素的值；最常用的方法}
for i:=1 to 10 do a[i]:=i;
```

{————数组元素 a[1]到 a[10]的值分别为 1 到 10；数据赋初值}

```
for i:=1 to 10 do a[i]:=0;
```

{————数组元素清 0；最常用的数据初始化的方法}

```
for i:=1 to 10 do a[i]:=random(100);
```

{————随机产生 10 个 100 以内的数，赋给各数组元素}

3、数组元素的输出：

和数组元素的输入相同，数组元素的输出也不能由一个 write 语句直接完成。同样要逐个数组元素输出。通常也用循环结构来完成这一功能：

```
for i:=1 to 10 do write(a[i],' ');{————数组元素之间用空格分隔}
writeln;
```

4、数组的应用：

例 1：从键盘输入 10 个数，将这 10 个数逆序输出，并求这 10 个数的和，输出这个和。

```
program p1;
var
    a:array [1..10] of integer;
    i,s:integer;
begin
```

```
    for i:=1 to 10 do read(a[i]);
    for i:=10 downto 1 do write(a[i],' ');
    writeln;
    s:=0;
    for i:=1 to 10 do s:=s+a[i];
    writeln('s=',s);
end.
```

例 2：从键盘输入 10 个数，将这 10 个数从大到小的顺序输出。

```
program p2;
Var n:array[1..10] of integer;
I,j,t:integer;
Begin
  For I:=1 to 10 do Readln(n[I]);
  For I:=1 to 9 do begin
  For j:=I+1 to 10 do begin
    If n[I]<n[j] then begin
      T:=n[I];
      N[I]:=n[j];
      N[j]:=t;
    End;
```

```
  End;
  End;
  For I:=1 to 10 do begin
  Write(n[I]:5);
  End;
End.
```

例 3：用筛法求 100 以内的素数（质数）。

分析：素数是除了 1 和它本身以外没有其它约数的数。用筛法求素数的方法是：用质数筛去合数：从第一个素数 2 开始，

把它的倍数去掉；这样 2 以后的第一个非 0 数就一定也是素数，把它的倍数也删了……重复这个删数过程，直到在所找到

的素数后再也找不到一个非 0 数。把所有非 0 数输出。

```
program p3;
var
    a:array [1..100] of integer;
    i,j,k:integer;
begin
     for i:=1 to 100 do a[i]:=i;
     a[1]:=0;i:=2;
```

```
    while i<=100 do
    begin
        k:=i;
        while k<=100 do
        begin
            k:=k+i;
            a[k]:=0;
        end;
        {————————上面将所有 a[i]的倍数清 0}
        i:=i+1;
        while a[i]=0 do i:=i+1;
        {————————查找接下来的第一个非 0 数}
    end;
    for i:=1 to 100 do if a[i]<>0 then write(a[i],' ');
end.
```

练习题：

1、随机产生 20 个 100 以内的数，输出；按从小到大的顺序排序，输出。

2、有一组数，其排列形式如下：11，19，9，12，5，20，1，18，4，16，6，10，15，2，17，3，14，7，13，8，

且尾部 8 和头部 11 首尾相连，构成环形的一组数，编程找出相邻的4

个数，其相加之和最大，并给出它们的起始位置。

第十四讲:二维数组

二维数组

〖语法分析〗

一维数组在编程中多用于描述线性的关系：如一组数；一组成绩；一组解答等。数组元素只有一个下标，表明该元素在数组中的位置。二维数组在编程中多数用于描述二维的关系：如地图、棋盘、城市街道、迷宫等等。而二维数组元素有两个下标：第一个下标表示该元素在第几行，第二个下标表示在第几列。二维数组的定义格式如下：

```
var
    a：array[1..10,1..5] of integer;
```

其中：a 是数组名，由程序员自定；array 和 of 是定义数组的保留字；（这两点和一维数组定义的格式一样）中括号中的两个范围表示二维数组共有多少行、多少列（第一个范围表示行数，第二个范围表示列数）；最后一个表示数组元素的类型，规定和一维数组一样。如上例，定义了一个二维数组 a，共有 10 行 5 列。

使用二维数组要注意：

1、数组元素的指称：数组名[行号，列号]。如第三行第四个元素：a[3,4]。

对某一行进行处理。如累加第 4 行的数据。则固定行号为 4。如：for i:=1 to 5 do s:=s+a[4,i];

对某一列进行处理。如累加第 4 列的数据。则固定列号为 4。如：for i:=1 to 10 do s:=s+a[i,4];

2、二维数组的输入输出要用双重循环来控制：

```
for i:=1 to 10 do{————————控制行数}
begin
    for j:=1 to 5 do read(a[i,j]){————————第一行读入 5 个元素}
    readln;{————————读入一个换行符}
end;
{————————最常用的方法：从键盘读入数据初始化二维数组}
for i:=1 to 10 do
    for j:=1 to 5 do a[i,j]:=0;
{————————最常用的方法：将二维数组清 0}
for i:=1 to 10 do
begin
    for j:=1 to 5 do write(a[i,j]:4);
    writeln;
end;
```

{————————最常用的输出方法：按矩阵形式输出二维数组的值}

3、二维数组的应用：

例 1：竞赛小组共有 20 位同学，这学期每位同学共参与了三项比赛，请统计每位同学的平均分。

分析：定义一个 20 行 3 列的二维数组来存放这些成绩。定义一个 20 个元素的一维数组来存放平均分。

```
program p1;
var
    a:array [1..20,1..3] of integer;
    b:array [1..20] of real;
    i,j:integer;
begin
    for i:=1 to 20 do
    begin
        for j:=1 to 3 do read(a[i,j]);
      readln;
    end;
{————————从键盘上读入 20 个同学的三次竞赛成绩}
    for i:=1 to 20 do b[i]:=0;
{————————先将平均分数组清 0}
    for i:=1 to 20 do
```

```
        begin
            for j:=1 to 3 do b[i]:=b[i]+a[i,j];{————————计算总分}
            b[i]:=b[i]/3;{————————计算平均分}
        end;
        for i:=1 to 20 do write(b[i]:5:1);
{————————输出平均分}
writeln;
end.
```

例 2：设有一程序:

```
        program ex5_3;
        const n=3;
        type matrix=array[1..n,1..n]of integer;
        var a:matrix;
           i,j:1..n;
        begin
           for i:=1 to n do
           begin
              for j:=1 to n do
                 read(a[i,j]);
              readln;
           end;
        for i:=1 to n do
```

```
begin
   for j:=1 to n do
      write(a[j,i]:5);
   writeln;
end;
end.
```

且运行程序时的输入为:

2□1□3←┘

3□3□1←┘

1□2□1←┘

则程序的输出应是:

2□3□1

1□3□2

3□1□1

例 3：输入一串字符,字符个数不超过 100,且以"."结束。 判断它们是否构成回文。

分析：所谓回文指从左到右和从右到左读一串字符的值是一样的，如 12321,ABCBA,AA 等。先读入要判断的一串字符（放入数组 letter 中），并记住这串字符的长度，然后首尾字符比较，并不断向中间靠拢，就可以判断出是否为回文。

源程序如下：

```
program ex5_5;
```

```
var
      letter     :   array[1..100]of char;
      i,j        :   0..100;
      ch         :   char;
begin
   {读入一个字符串以'.'号结束}
   write('Input a string:');
   i:=0;read(ch);
   while ch<>'.' do
   begin
      i:=i+1;letter[i]:=ch;
      read(ch)
   end;
   {判断它是否是回文}
   j:=1;
   while (j<i)and(letter[j]=letter[i])do
   begin
      i:=i-1;j:=j+1;
   end;
   if j>=i   then writeln('Yes.')
   else writeln('No.');
end.
```

例 4：奇数阶魔阵

魔阵是用自然数 1，2，3…，n2 填 n 阶方阵的各个元素位置，使方阵的每行的元素之和、每列元素之和及主对角线元素之和均相等。奇数阶魔阵的一个算法是将自然数数列从方阵的中间一行最后一个位置排起，每次总是向右下角排（即 Aij 的下一个是 Ai+1,j+1）。但若遇以下四种情形，则应修正排数法。

（1） 列排完（即 j=n+1 时），则转排第一列;

（2） 行排完（即 i=n+1 时），则转排第一行;

（3） 对 An,n 的下一个总是 An,n-1;

（4） 若 Aij 已排进一个自然数，则排 Ai－1，j－2。

例如 3 阶方阵，则按上述算法可排成：

4 3 8

9 5 1

2 7 6

有了以上的算法，解题主要思路可用伪代码描述如下：

```
1  i←n div 2+1,y←n /*排数的初始位置*/
2  a[i,j]←1;
3  for k:=2 to n*n do
4  计算下一个排数位置←（i,j）；
5  if a[i,j]<>0 then
6  i←i-1;
```

```
7    j←j-2;
8    a[i,j]←k;
9    endfor
```

对于计算下一个排数位置，按上述的四种情形进行，但我们应先处理第三处情况。算法描述如下：

```
1    if (i=n)and(j=n) then
2    j←j-1; /*下一个位置为(n,n-1)*/;
3    else
4    i←i mod n +1;
5    j←j mod n +1;
6    endif;
```

源程序如下：

```
program ex5_7;
var
   a : array[1..99,1..99]of integer;
   i,j,k,n : integer;
begin
   fillchar(a,sizeof(a),0);
   write('n=');readln(n);
   i:=n div 2+1;j:=n;
```

```
a[i,j]:=1;
for k:=2 to n*n do
   begin
      if (i=n)and(j=n) then
         j:=j-1
      else
         begin
            i:=i mod n +1;
            j:=j mod n +1;
         end;
      if a[i,j]<>0 then
         begin
            i:=i-1;
            j:=j-2;
         end;
      a[i,j]:=k;
   end;
for i:=1 to n do
   begin
      for j:=1 to n do
         write(a[i,j]:5);
      writeln;
```

```
        end;
    end.
```

练习题：

1、 输入 N 个同学的语、数、英三科成绩，计算他们的总分与平均分，并统计出每个同学的名次，最后以表格的形式输出。

2、输入 10 个学生的姓名，编一程序将它们按字母的顺序排列。

3、 输出杨辉三角的前 N 行(N<10)。

```
1
1  1
1  2  1
1  3  3  1
1  4  6  4  1
```

4、求一个 5 X 5 数阵中的马鞍数，输出它的位置。所谓马鞍数，是指在行上最小而在列上最大的数。如下：

5 6 7 8 9

4 5 6 7 8

3 4 5 2 1

2 3 4 9 0

1 2 5 4 8

则 1 行 1 列上的数就是马鞍数。

5、验证数学黑洞：所有四位数，除了数字全相同的外，其它的，经过不多于七次的下列操作，一定可以得到6174；并且，一旦得到6174之后，就掉进黑洞，再也得不到其它的数（6174：7641-1467 = 6174）：将这个四位数的数字按从大到小和从小到大重组成两个数，大数减去小数。

例：输入3214

（1）4321 - 1234 = 3087

（2）8730 - 378 = 8352

（3）8532 - 2358 = 6174

6、猴子选大王：有M个猴子围成一圈，每个有一个编号，编号从1到M。打算从中选出一个大王。经过协商，决定选大王的规则如下：从第一个开始，每隔N个，数到的猴子出圈，最后剩下来的就是大王。

要求：从键盘输入M，N，编程计算哪一个编号的猴子成为大王。

7、有M个人围成一圈，每人有一个编号，从编号为1的人开始，每隔N个出圈，按出圈次序排成一列，其编号刚好按顺序从1到M。

要求：从键盘输入M，N，编程计算并输出这M个人原来在圈中的位置。

8、前N个自然数排成一串: X1,X2,X3.....Xn，先取出x1,将x2,x3移到数串尾,再取出x4,将x4,x6移到数串尾,..... 类推直至取完.取出的序列恰好是:1,2,3.....n，要求输入N,求原来的数串的排列方式.

9、猴子捉兔子：围绕着山顶有１０个洞，狐狸要吃兔子，兔子说：“可以，但必须找到我，???? ???我就藏身于这十个洞中，你从１０号洞出发，先到1号洞找，第二次隔1个洞找，第三次隔2个洞找，以后如此类推，次数不限。”但狐狸从早到晚进进出出了１０００次，仍没有找到兔子。问兔子究竟藏在哪个洞里？

第十五讲:字符与字符串处理

字符与字符串处理

一、字符与字符串类型的使用

Var 字符变量名：char；

字符类型是一个有序类型，字符的大小顺序按ASCII码的大小决定。相关的函数有succ、pred、ord、chr等。

```
VAR A:CHAR;
BEGIN
READLN(A);
WRITELN('A:',A);
```

```
WRITELN('SUCC(A):',SUCC(A));

WRITELN('PRED(A):',PRED(A));

WRITELN('ORD(A):',ORD(A));

WRITELN('CHR(ORD(A)):',CHR(ORD(A)));

END.
```

例：按字母表顺序和逆序每隔一个字母打印。即打印出：

a c e g i k m o q s u w y

z x r v t p n l j h f d b

```
for i:=1 to 13 do write(chr(63+2*i):4);writeln;

for i:=1 to 13 do write(chr(92-2*i):4);writeln
```

字符串

〖语法分析〗

字符串用于存放整批的字符数据。通常编程中使用字符串存放字符化了的数字数据。如高精度运算时存放操作数和运算结果。字符串可以看作是特殊的字符串数组来处理。当然，它也有自己的特点。下面是字符串定义的格式：

```
var s:string; s1:string[15];      s[1] s[2] … s[255]      s[0]      ord(s[0])
```

字符串定义时，如不指定长度，则按该类型的最大长度（255 个字符）分配空间，使用时最大可用长度为 255 个；如果在中括号中给出一个具体的值（1—255 之间），则按这个值的大小分配空间。使用时，最大的可用长度即为该值。

1、字符串的输入、输出：

字符串类型既可按数组方式输入、输出，也可直接输入、输出：readln(s)；writeln(s)；多个字符串输入时以回车作为数据间的分隔符；每个 readln 语句只能读入一个字符串。

2、有关字符串的操作：

操作	类型	作用	返回值	例子
length(s)	函数	求字符串 s 的长度	整型	s:='123456789'; l:=length(s);{l 的值为 9} 字符串的长度存放在 s[0]中，ord(s[0])的值与 length(s)的值相同。
copy（ s,w,k)	函数	复制 s 中从 w 开始的 k 位	字符串	s:='123456789'; s1:=copy(s,3,5);{s1 的值是 '34567'}

val(s,k,code)	过程	将字符串 s 转为数值，存在 k 中；code 是错误代码		var s:string;k,code:integer;
				begin
				s:='1234';
				val(s,k,code);
				write(k);{k=1234}

str(i,s)	过程	将数值 i 转为字符串 s		i:=1234;
				str(i,s);
				write(s);{s='1234'}

Delete(s,w,k) 过程	在 s 中删除从第 w 位开始的 k 个字符		s := 'Honest Abe Lincoln'; Delete(s,8,4); Writeln(s); { 'Honest Lincoln' }	

Insert(s1, S, w)	过程	将 s1 插到 s 中第 w 位		S := 'Honest Lincoln'; Insert('Abe ', S, 8); { 'Honest Abe Lincoln' }
Pos(c, S)	函数	求字符 c 在 s 中的位置	整型	S := ' 123.5'; i :=Pos(' ', S);{i 的值为 1}
+	运算符	将两个字符串连接起来		s1:='1234'; s2:='5678'; s:=s1+s2;{'12345678'}

[例 6、3]把 26 个英语字母正向、逆向打印出来。

```
Const s:string[26]=’abcdefghijklmnopqrstuvwxyz’;
Var t:string[26];
I:integer;
Begin
  t:=’                          ‘;          {共 26 个空格}
  For I:=1 to 26 do begin
     T[I]:=s[27-I];
  End;
  Writeln(s);
  Writeln(t);
```

End.

[例 6、4]找出所有的四位回文数：（回文数就是一个数从左往右读与从右往左读都是同一个数）

```
var s:string[4];
n:integer;
begin
  for n:=1000 to 9999 do begin
      str(n,s);
      if (s[1]=s[4]) and (s[2]=s[3]) then write(n:6);
  end;
end.
```

或者用如下程序：

```
var n:integer;
s,t:string;
begin
  for n:=10 to 99 do begin
      str(n,s);
      t:=s+s[2]+s[1];
write(s:6);
  end;
end.
```

上述两个程序，哪个快，哪个慢？

练习题：

1、读入一串字符，以句号结束，然后让其倒序输出。

如输入：I am a student.

输出：tneduts a ma I

2、读入一串数字，以句号结束，请统计其中‘0’到‘9’的各个数字的个数。

如输入：91254782354987012345978.

输出：0:1 1:2 2:3 3:2 4:3 5:3 7:3 8:3 9:3

3、输入一段文章（255 个字符以内），求文章中单词的个数（相同单词只记一次，The 和 the 认为是同一个单词，只记一次）。

4、请编写一个程序，让它能够计算两个 200 位以内的整数的和。

5、做一个加法器。完成 30000 以内的加法，两个加数间用“+”连接，可以连加，回车表示式子输入完成；“#”表示结束运算，退出加法器。

看程序写结果

1.

```
var ch:string;
        i:byte;
begin
    readln(ch);
```

```
    for i:=1 to ord(ch[0]) do
        write(ch[i]:2);
    writeln
end.
```

输入：My name is Tom.

输出：

2.

```
var i:integer;
begin
    for i:=1 to 13 do write(chr(63+2*i):4);
    writeln;
    for i:=1 to 13 do write(chr(92-2*i):4);
    writeln
end.
```

输出：

3.

```
var a:char;
begin
    readln(a);
    writeln('a:',a);
    writeln('succ(a):',succ(a));
```

```
    writeln('pred(a):',pred(a));
    writeln('ord(a):',ord(a));
    writeln('chr(ord(a)):',chr(ord(a)));
    writeln('chr(ord(a)+2):',chr(ord(a)+2));
end.
```

输入：e

输出：

4.

```
var s:integer;
ch:char;
        count:array['a'..'z'] of integer;
begin
    for ch:='a' to 'z' do count[ch]:=0;
       read(ch);
    while ch<>'!' do
    begin
       if (ch>='a')and(ch<='z') then
          count[ch]:=count[ch]+1;
       read(ch)
    end;
    s:=0;
    for ch:='b' to 'z' do s:=s+count[ch];
```

```
    writeln(s)
end.
```

输入：We are proud OF OUR COUNTRY!

输出：

5.

```
var a:array[1..100] of string[20];
        s:string;temp:string[20];
        i,j,k,n:integer;
begin
    readln(n);readln(s);i:=1;j:=0;k:=0;
    while i<=length(s) do
       if s[i] in ['0'..'9'] then
       begin
          j:=1;
          while s[i] in ['0'..'9'] do
          begin
             temp[j]:=s[i];
             inc(j);inc(i) {inc(i)相当于 i:=i+1}
          end;
          temp[0]:=chr(j-1);
          k:=k+1;
```

```
            a[k]:=temp
          end
      else
      while s[i]=' ' do inc(i);
      for i:=1 to n-1 do
          for j:=i+1 to n do
              if (a[i]+a[j])<(a[j]+a[i]) then
                  begin
                      temp:=a[i];a[i]:=a[j];
                      a[j]:=temp
              end;
      for i:=1 to n do write(a[i]);
      writeln
end.
```

输入：4

81　792　39　79

输出：

综合练习一(顺序结构、分支结构、循环结构、数组)

说明：3 小时内完成。

1、（30 分）输入年、月、日，求这一天是星期几。

算法提要：求出这一天离公元 1 年的元旦有多少天，然后对 7 求余{公元 1 年的元旦为星期一}

2、（30 分）键入自然数 N，打印出一个上下左右全方位对称的字符图形，其形如田字，边长 2*N+1 个字符，笔划均按字母顺序递增或递减对称排列。

例如：N=3 时，图形为：

ABCDCBA

B? ? C? ? B

C? ? B? ? C

DCBABCD

V? ? B? ? C

B? ? C? ? B

ABCDCBA

3、（40 分）键入两个自然数 N 和 K，将 N 写成 K 个大于 1 的自然数相乘，要求这 K 个数按从小到大排列，而且除了第 K 个数之外，前面（K-1）个数是 N 分解出来的最小自然数。

例如：N=24，K=2 时，输出为 24=2*12，而不是 24=4*6

N=3，K=2 时，输出则为“No answer！”

第十六讲:函数

函数

PASCAL 给我们提供了一些标准函数，我们不用了解这些函数如何求出来的，只管直接调用它们，挺方便的。如正弦函数，余弦函数，算术平方根．．．．．．有了这些函数，我们觉得很省事。如：求 SIN（1）＋SIN（2）＋．．．＋SIN（100）＝？这个程序我们可以这样编写：

例 1

```
PROGRAM    e1(input,output);
 VAR i:integer;
     s:real;
 BEGIN
     s:=0;
     for i:=1 to 100 do
       s:=s+sin(i);
     writeln('s=',s);
 END.
```

在这个程序里，我们直接用到了正弦函数，至于 SIN（1），SIN（2）如何求出来的我们不需过问，只管直接用它的结果便是了。

我们来看看下面一个例子：求：1！＋2！＋3！＋．．．＋10！＝？

如果要编写程序，我们看到求阶乘的操作要执行 10 次，只不过每次所求的数不同。我们想：不至于编写 10 遍求阶乘的程序吧。我们希望有一个求阶乘的函数，假设为 JS（X），那么我们就可以这样求这道题了：

例 2

```
PROGRAM e1(input,output);
  VAR i:integer;
      s:real;
  BEGIN
      s:=0;
      for i:=1 to 10 do
        s:=s+js(i);
      writeln('s=',s);
  END.
```

现在的问题是：TURBO PASCAL 没提供 JS（X）这样一个标准函数，这个程序是通不过的。如果是 PASCAL 的标准函数，我们可以直接调用，如 TRUNC（X），LN（X），SQRT（X）．．．．．．而 PASCAL 提供给我们的可供直接调用的标准函数不多。没关系，我们编写自己的函数！

- **函数的编写**

自己编写一个函数，它的格式如下：

FUNCTION 函数名（形式参数表）：函数类型；

VAR 函数的变量说明；

BEGIN

函数体

END；

我们来分析一下，一个函数的编写可分成三部份：一是函数首部，即第一个语句。它必须以 FUNCTION 开头，函数名是自己取的，取名的原则是便于记忆，和文件名的取名规则类似。形式参数（简称形参）表以标识符的形式给出，相当于函数中的自变量。参数可以有多个，也可以有多种类型。不同类型的参数之间用“；”隔开，同类型的参数如有多个，则用“，”隔开，在其后得加上说明。如：

FUNCTION A1（A，B，C：INTEGER；D，E，F：REAL）：REAL；

在最后，函数属于哪种类型，还得表示出来。在本例中，该函数为实型。

第二部分是函数的变量说明部分，对在本函数中将要用到的变量作类型说明，这一点和以前学的变量一样。如果程序不用变量，则此部分也可省掉。

第三部分是函数体，本函数的功能实现就在于此，编写的语句就在里面。

例 3 编写一求阶乘的函数。

我们给此函数取一名字就叫 JS。

```
fUNCTION js(n:integer):longint;
   var i:integer;
     s:longint;
   begin
     s:=1;
     for i:=1 to n do
       s:=s*i;
 js:=s;
```

```
end;
```

在本例中，函数名叫 JS，只有一个 INTEGER 型的自变量 N，函数 JS 属 LONGINT 型。在本函数中，要用到两个变量 I，S，在 VAR 后已加以说明。在函数体中，是一个求阶乘的语句，但有一点要注意：虽然 N 的阶乘的值在 S 中，但最后必须将此值赋给函数 JS，此时 JS 不带任何参数。在任何函数中，最后都要把最终结果赋给函数名，因为该函数的结果是靠函数名返回的。

在这里，函数的参数 N 是一个接口参数，说得更明确点是入口参数。如果我们调用函数：JS（3），那么在程序里所有有 N 的地方 N 被替代成 3 来计算。在这里，3 就被称为值参。又如：SQRT（4），LN（5），这里的 4，5 叫值参。而 SQRT（X），LN（Y）中的 X，Y 叫形参。

- **函数的调用**

自定义的函数在调用前要先说明，在主程序中的位置如下:

PROGRAM 程序名（INPUT，OUTPUT）；

VAR 主程序变量说明；

FOUNCTION 函数名（形参表）：函数类型；

VAR 函数变量说明；

BEGIN

函数体

END；{FUNCTION}

BEGIN

主程序

END .{PROGRM}

在主程序中，我们把函数的全部说明放在主程序的变量说明和程序体之间，然后在主程序的执行部分就可以直接调用自定义函数了。注意：在函数的说明部分，我们要用形参，但在程序的执行部分调用自定义函数时，就得用值参了。

例 4 利用前面定义的阶乘函数，求 5！，9！。

```
PROGRAM e59(input,outout);
  VAR a1,a2:longint;
fUNCTION js(n:integer):longint;
  var i:integer;
      s:longint;
  begin
```

```
    s:=1;
    for i:=1 to n do
      s:=s*i;
    js:=s;
  end;
BEGIN
    a1:=js(5);
    a2:=js(9);
    writeln('5!=',a1,' ','9!=',a2);
END.
```

在这个程序中，在主程序的 BEGIN 之前，我们对函数进行了一次说明，在后面的程序中都可以象标准函数那样直接调用自定义函数了。在 FUNCTION 语句中，用的是形参 N，在主程序调用中，调用函数是用的值参，如：JS（5）；程序执行到这儿会自动将 5 代入前面的 FUNCTION 函数中，用 5 取代所有的 N，最终将结果赋值给 JS。所以在 A1 中一定是 5！，A2 中是 9！。另外，函数不能单独使用，一定要结合主程序才能运行。

如果是求 1！＋2！＋3！＋．．．＋10！，则只需把主程序改成：

A1：＝0；

FOR J：＝1 TO 10 DO

A1：＝A1＋JS（J）；

WRITELN（A1）；

在例 4 中，主程序的变量 A1，A2 叫全程变量，它们除了主程序外，还可以在函数中出现；在函数说明中用到的变量 I，S 则是局部变量，只能在函数部分使用，一旦出了函数则失去意义；别外要注意：全程变量和局部变量尽量不要同名。

例 5 任意输入 10 组三角形的三边，求其面积。

已知三角形的三边，是可以求出面积的。我们可以定义一个已知三角形三边求其面积的函数，设为 AREA（a1,a2,a3）。

```
PROGRAM e5(inmput,output);
   VAR a,b,c,s:real;
       i:integer;
   FUNCTION    area(a1,a2,a3:real):real;
     var s1,d:real;
     begin
     d:=(a1+a2+a3)/2;
     s1:=Sqrt(d*(d-a1)*(d-a2)*(d-a3));
```

```
    area:=s1;
    end;

    BEGIN
    for i:=1 to 10 do
      begin
        writeln('input a,b,c');
        readln(a,b,c);
        if   (a+b<=c) or (a+c<=b) or(b+c<=a)
          then writeln('data error!')
            else writeln('s=',area(a,b,c));
      end;
  END.
```

在函数说明中，如果形参的个数不止一个，那么在程序中调用函数的实参个数一定要与形参的个数一致，第一个实参对应第一个形参，第二个实参对应第二个形参．．．．．．次序不能调。

例 6 定义一个函数 CHECK（N，D），让它返回一个布尔值。如果数字 D 在整数 N 的某位中出现则送回 TRUE，否则送回 FALSE。

例如：CHECK（325719，3）＝TRUE；

CHECK（77829，1）＝FALSE；

```
PROGRAM e6(input,output);
 VAR a,b:integer;


  FUNCTION
check(n,d:integer):boolean;
  var f:boolean;
    e:integer;
  begin
    f:=false;
    while (n>0) and (not f) do
        begin
          e:=n mod 10;
          n:=n div 10;
          if e=d then f:=true;
    end;
    check:=f;
```

```
end;

BEGIN
writeln('input n,d');
read(a,b);
writeln(check(a,b));
END.
```

练习：编写一个函数，返回布尔值判别输入三个字符是按序或无序排列。程序中若输入 abc 则显示顺序排列，若输入 cba 则显示逆序排列，其他情况显示无序排列，当输入***程序结束。

第十七讲:过程

过程

过程和函数一样，也是子程序。一个过程对应一个需要完成的任务。PASCAL 中提供了不少标准过程，如：READ，WRITE，GET，NEW，PUT．．．．．这些标准过程在程序中可以直接调用。但仅仅这些标准过程还不能满足我们的需要，我们还要自己定义过程，就象函数一样。但函数必须以值的形式返回，而过程不一定返回一个值，

只是执行一个任务而已；函数只能返回一个值，而过程可以返回不止一个值。所以函数不能取代过程。

2．2　过程的定义

过程定义的格式如下：

```
PROCEDURE    过程名（形式参数）；
    VAR    过程的变量说明；
    BEGIN
      过程体
    END;
```

对该格式说明如下：

一个过程也分为三部分，1：过程的首部。过程必须以 PROCEDURE 开头，过程名的取名规则和函数名一样，括号里面是形式参数，如形参不止一种，则中间用“；”隔开，同类形参如不止一个，则中间用“，”隔开。另：有时侯过程不用加参数。2：过程的说明部分，用 VAR 开头，它只能对过程中的变量进行说明，同样是局部变量。另：如果过程不用变量，则可将说明部分省略。3：过程体。它是过程的执行部分。

我们来定义一个打印由“*”组成的矩阵的过程，该矩阵四行五列。

例 7

```
PROCEDURE print;
    var i,j:integer;
    begin
      for i:=1 to 4 do
        begin
          for j:=1 to 5 do
            write('*');
            writeln;
          end;
    end;
```

该过程就没有参数，直接执行打印一个固定矩阵的任务，而且也没返回值。

例 7　　定义一个求 N! 的过程。

```
PROCEDURE   js(n:integer);
  var s:longint;
```

```
        i:integer;
begin
    s:=1;
    for i:=1 to n do
      s:=s*i;
    writeln(n,'!=',s);
end.
```

在该过程中，它的值的返回形式和函数不一样：函数是由函数名返回，而过程不是由函数名返回的；在过程的首部不用对过程的类型进行说明。

2．3　过程的调用

自定义过程在程序调用之前要先说明，过程的说明就在主程序的执行语句之前。其格式如下：

PROGRAM　程序名（INPUT，OUTPUT）；

VAR　主程序的变量说明；

PROCEDURE　过程名（形式参数表

功；

VAR　过程的变量说明；

BEGIN

过程体

END;

BEGIN

主程序体

END.

例 8　　任意输入 10 个三角形的三边，用过程把其面积求出。

我们把求一个已知三边的三角形的面积定义一个过程，对比一下和例 5 有什么不同。

PROGRAM

e5(inmput,output);

VAR a,b,c,s:real;

i:integer;

```
                                                    PROCEDURE
area(a1,a2,a3:real);
                                                        var
s1,d:real;
                                                        begin

d:=(a1+a2+a3)/2;

s1:=Sqrt(d*(d-a1)*(d-a2)*(d-a3));

writeln('s=',s1);
                                                          end;

                                                   BEGIN
                                                      for i:=1
to 10 do
                                                      begin
```

```
writeln('input a,b,c');

readln(a,b,c);
                                                    if
(a+b<=c) and (a+c<=b) and (b+c<=a)

then writeln('data error!')

else area(a,b,c);
                                                    end;
                                                  END.
```

我们看到：过程的调用和函数不同。函数不能作为独立的一个语句使用，而过程可以。函数的值是由函数名返回的，而过程不能。

现在我们提出一个要求：用过程求出 1！＋2！＋3！＋．．．＋10！＝？

求 N！的问题我们在例 7 已写出来，但阶乘的结果是在过程里用 WRITE 语句输出的，不能用累加语句把结果求出来。那么，怎样用过程将类似的问题求出来？这就得用到变量参数。

2．4　变量参数

在过程定义的语句中，有个参数表，在参数表中，除了前面我们已用的值参，还有变量参数。变量参数的作用是：它可以作为过程的出口参数。我们可以把过程中求出的结果用变量参数输出到过程外，在过程外面可以调用该参数，因此，该参数是全局变量。其格式上的区别是在变量参数前加上VAR即可。

那么我们现在来求1！＋2！＋3！＋．．．＋10！＝？

例9

```
PROGRAM    e9(input,output);
VAR j:integer;
        s,m:longint;

PROCEDURE js(n:integer;var
m:longint);
    var i:integer;
    begin
        m:=1;
        for i:=1 to n do
            m:=m*i;
```

```
end;

BEGIN
    s:=0;
    for j:=1 to 10 do
        begin
            js(j,m);
            s:=s+m;
        end;
    writeln('s=',s);
END.
```

在本例中。我们看到，过程 JS 中用到了变量参数 M，在过程中定义为 LONGINT 类型；而在主程序的变量说明中也得对变量参数 M 说明为同种类型 LONGINGT。于是，在过程中和主程序中都可以用该变量了。我们在调用这类过程时，在括号中要写上所有的参数，但用不着写上参数的类型和 VAR 了。

练习：

1、某部队举行一次军事演习，A、B 两队约好在同一时间从相距 100 公里的各自驻地出发相向运动。A 队的速度为 10 公里/小时，B 队的速度为 8 公里/小时。一通讯员骑马从 A 地同时出发为行进中的两队传递消息，速度为 60 公里/小时。每遇一队立即折回驶向另一队，当两队距离小于 0.5 公里时，停下来不再传递消息。求此时通讯员跑了多少趟(从一队到另一队为一趟)？

2、已知二个合数 A=18 和 B=96，键入 N 个[10,400]之间的自然数，求这 N 个数中所有合数与 A、B 的最大公约数。

第十八讲:子程序中的参数

- 形参和实参

子程序调用(过程调用或函数调用)的执行顺序分为以下几步：

实参和形参结合——〉执行子程序——〉返回调用处继续执行

子程序说明的形式参数表对子程序体直接引用的变量进行说明，详细指明这些参数的类别、数据类型要求和参数的个数。子程序被调用时必须为它的每个形参提供一个实参，按参数的位置顺序一一对应，每个实参必须满足对应形参的要求

Turbo Pascal 子程序形参有四类：

1.值参数

形式参数表中前面没有 var，后有类型的参数。它类似过程和函数的局部变量，仅为过程和函数的执行提供初值而不影响调用时实际参数的值。在调用过程或应用函数时，值参数所对应的实际参数必须是表达式，而且它的值不能使文件类型或包括文件类型的值。实参必须和形参赋值相容。

2.变量参数

形式参数表中前面有 var 后由类型的参数。如果需要子程序向调用程序返回值时，应采用变量参数。变量参数要求它的实参是和它同一类型的变量。因为在子程序执行时，遇到对相应形参的引用式定值，就是对相应实参的引用式定值，即对形参的任何操作就是对实参本身的操作。

3.无类型变量参数

形式参数表中前面有 var 而后面没有类型的参数。形参是无类型变量，对应的实参允许为任意类型的变量，但要在子程序中设置的强制类型转换（类型名（无类型变量参数名）），将无类型变量参数改变为相应类型。

4.子程序参数

用过程首部或函数首部作为形式参数。

- 标识符的作用域

1.全局变量和它的作用域

全局变量是指在程序开头的说明部分定义和说明的量。它的作用域分为两种情况：

(1)在全局变量和局部变量不同名时，其作用域是整个程序。

(2)在全局变量和局部变量同名时，全局变量的作用域不包含同名局部变量的作用域。

2.局部变量和它的作用域

凡是在子程序内部使用的变量，必须在子程序中加入说明。这种在子程序内部说明的变量称为局部变量。局部变量的作用域是其所在的子程序。形式参数也只能在子程序中有效。因此也属于局部变量。局部变量的作用域分为两种情况：

(1)当外层过程序的局部变量名和嵌套过程中的局部变量不同名时，外层过程的局部变量作用域包含嵌套过琛。

(2)当外层过程的局部变量名和嵌套过程内的局部变量名同名时，外层局部变量名的作用域不包含此过程。

作用域的示例:

```
program ppro4;
var
  a:integer;
procedure add(x:integer);
  var
```

```
    a1:integer;
  begin
    a1:=x+10;
    writeln('a1=',a1);
  end;
begin
  a:=15;
  add(a);
  writeln('a=',a);
end.
```

==========

```
program ppro5;
var
  a:integer;
procedure add(var c:integer);
  begin
    c:=c+10;
    writeln('c=',c);
  end;
begin
  a:=10;
```

```
    add(a);
    writeln('a=',a);
  end.
```

第十九讲:集合与记录

集合

一.集合的定义：

type 类型名=set of 基类型

例如：

```
type
  num=set of char;
var
  n:num;
```

或

```
var
  n: set of char;
```

二.集合的运算：

1.集合的表示：

用一组方括号括号一组元素来表示，元素之间用逗号分隔。如：

[A,B,C,D]－－有四个枚举量的集合

['A','B','C','D']－－有四个字符的集合

[1..20]－－包含了 1 到 20 中所有整数的集合

[0]－－只有一个元素 0 的单元素集

[]－－空集

2.集合的运算：

(1)赋值(:=):

ll:=['A'..'C'];

(2)并(+):

[0..7]+[0..4]的值为[0..7]

(3)交(*):

[0..7]*[0..4]的值为[0..4]

(4)差(-):

[0..7]-[0..4]的值为[5..7]

(5)相等(=):

[0..7]=[0..4]的值为 false

(6)不等(<>):

[0..7]<>[0..4]的值为 true

(7)包含于(<=):

[0..7]<=[0..4]的值为 false

(8)包含(>=):

[0..7]>=[0..4]的值为 true

(9)成员存在于 (in):

1 in [0..4]的值为 true

3.注意：

(1)集合运算相当快，在程序中常用集合表达式来描述复杂的测试。如

A)条件表达式： (ch='T') or (ch='t') or (ch='Y') or (ch='y') 可用集合表达式表示为：

ch in ['T','t','Y','y']

B)if (ch>=20) and (ch<=50) then ...;

可写成：

if ch in [20..50] then ...;

(2)集合类型是一种使用简便，节省内存面又运算速度快的数据类型。

(3)Turbo Pascal 规定集合的元素个数不超过 256 个(当实际问题所需的元素个数大于 256 时， 可采用布尔数组代替集合类型)。所以如下定义是错误的： var i: set of integer;

(4)集合类型变量不能进行算术运算，了不允许用读/写语句直接输入/输出集合。 所以集合的建立：

A)要通过赋值语句实现;

B)或先初始化一个集合，然后通过并运算向集合中逐步加入各个元素.

(5)集合元素是无序的，所以 ord，pred 和 succ 函数不能用

于集合类型的变量。

4.例 1：建立两个集合，然后对它们进行多种集合运算，并且输出结果。

5.例 2：运用集合完成筛法找素数。

三、练习：

1、从键盘输入一个字符串，进行下面的变换：

(1)数字 0,1,2,3,...,9 分别和字母 a,b,c,...,j 互换；

(2)字母 k,m,p,t,y 分别与其后继互换；

(3)其它字母和空格保持不变，输入字符串以句号结束；

2.编程读入两个字符串，然后输出如下信息：

(1)出现在某一个字符串中至少一次的字母和数字；

(2)同时出现在两个字符串中至少一次的字母和数字；

(3)出现在一个字符串中而不出现在另一个字符串中的字母和数字；

(4)不出现在任何字符串中的字母和数字。

记 录

一.记录的定义：

type 类型标识符=record

字段名 1：类型 1；

字段名 2：类型 2；

...

字段名 n：类型 n；

end;

如：

```
type
    studata=record
        num:string[6];
        name:string[8];
        sex:boolean;
        s:array[1..5] of real;
    end;
var
    student:studata;
    students:array[1..10] of studata;
```

二.记录的运用：

1.对记录中和个域的引用，要写出记录名和域名，如：student.num

2.开域语句：with。

with 记录名 do 语句；

或

with 记录名 1，记录名 2，... do 语句；

注意：

(1) 在 do 后面语句中使用的记录的域时，只要简单地写出域名就可以了， 域名前的记录变量和"."均可省略。

(2) 在关键字 with 后面，语句可以是一个简单语句，了可以是一个复合语句。

(3) 虽然在 with 后可以有多个记录变量名，但一般在 with 后只使用一个记录变量名。

[例 6、2]编一程序，用来记录 20 个学生的姓名、年级、班级、性别、语文成绩、数学成绩、总分，平均分。

```
Type student=record
    Name:string;
    Grade,class,chinese,mathe,sum:integer;
    Average:real;
    Male:boolean;
End;
Var s:array[1..20] of student;
    I,j:integer;
Begin
  For I:=1 to 20 do begin
    Write(‘please input the student’s
```

```
name,grade,class,male,chinese,mathe);
        Readln(s[I].name);
        Readln(s[I].grade);
        Readln(s[I].class);
        Readln(s[I].male);
        Readln(s[I].chinese);
        Readln(s[I].mathe);
        S[I].sum:=s[I].chinese+s[I].mathe;
        S[I].average:=s[i].sum/2
    End;
    ……
    {如果程序要求打印或排序，只需在此增加这些功能语句}
end.
```

相关评论

流云于 2005-5-9 8:04:56

关于集合和记录

集合是一序列有共同性质或联系的数据组成的一个集体。如日常生活中所说的：一个班的全体同学；一个书架上所有的书等。而在 PASCAL 语言中，集合一般都是指一系列的数值或字符等数据，如：小于 10 的正整数；26 个英语字母等。集合也象常 /变量一样有自己的名字，即集合名，它是在程序头部象定义变量一

样来定义的。

记录型数据类型是 PASCAL 语言中做搜索问题中一定要用到的数据类型，所以，它在 PASCAL 语言中占着非常重要的地位。学过数据库管理软件的人都知道，在数据库文件中，有一个非常重要的概念 —记录，例如：一个人的各门成绩，一个人的各种信息（姓名、性别等等），我们都把它们存放成一条记录而给予一个统一的名称。

第二十讲:文件

文件是一种构造型的数据类型。在程序中都需要产生一些输出，也需要接受若干个输入。这些输入、输出实际上是用文件的方法来实现的，在 Pascal 中用标准文件“input”和“output”来实现，它们分别对应标准输入设备和标准输出设备（可省略不写）这也就是一些程序的程序书写如下的原因了：

```
program ex(input,output);
...
```

但有时大量数据的读入和输出都是来自磁盘文件，这就要求我们必须熟练掌握对磁盘文件的操作。

对于我们来说，我们只必须掌握文本文件（或称正文文件，text）的读写即可:

1.文本文件的定义：

文本文件不是简单地由某类型的元素序列所组成，它的基本元素是字符，由它们构成行，若干行组成一份原文。由于各行的长度可以不同，所以文本文件只能顺序地处理。文本文件的定义如下：

```
var
  fp:text;
```

2.文本文件的读操作：

(1)调用 assign 过程，把磁盘文件赋予文本文件变量；

```
assign(fp,filename);
```

(2)调用 reset 过程，为读操作做准备；

```
reset(fp);
```

(3)在需要读数据的位置调用 read 过程或 readln 过程。

```
readln(fp,var1,var2,...,varn);
```

3.文本文件的写操作：

(1)调用 assign 过程，把磁盘文件赋予文本文件变量；

```
assign(fp,filename);
```

(2)调用 rewrite 过程，为写操作做准备；

```
rewrite(fp);
```

(3)在需要写数据的位置调用 write 过程或 writeln 过程。

```
writeln(fp,var1,var2,...,varn);
```

4.文本文件的关闭操作：

close(fp);

5.文本文件的其他操作：

(1)EOF(fp)—布尔函数，用于判断文件结束否。

(2)EOLN(fp)—布尔函数，用于判断行结束否。

[例 8、1]把 1 到 100 这 100 个数字写到一个文本文件（ANSWER.TXT）中。

```
Var I:integer;
   F:test;
Begin
   Assign(f,'answer.txt');
   Rewrite(f);
   For I:=1 to 100 do writeln(f,I);
   Close(f);
End.
```

[例 8、2]把以下文本文件中的十个整数读出来，然后按大小顺序写回此文件中。

NUMBER.TXT

3 5 7 1

2 8 12 54 14 9

程序如下：

```
var f:text;
n:array[1..10] of integer;
   I,j:integer;
begin
   assign(f,'number.txt');
   reset(f);
   read(f,n[1],n[2],n[3],n[4]);
   readln(f);
   readln(f,n[5],n[6],n[7],n[8],n[9],n[10]);
   close(f);
   {此处语句为排序数组 N，大家自己加入}；
   rewrite(f);
   for I:=1 to 10 do writeln(f,I);
   close(f);
end.
```

练习：

1.编写程序从磁盘上读取一个由 100 个实数组成的实型数据文件(indata.dat)，以此文件中所有大于平均值的实数建立一个名为“above.dat”的文件，其余的建立一个名为“rest.dat”的文件。

2.写一个程序，把文本文件中所有 GOOD 改为 BAD。

第二十一讲:指针

- 指针的动态变量

1.定义指针类型

在 Turbo Pascal 中，指针变量中存放的某个存储单元的地址，即指针变量指向某个存储单元。一个指针变量仅能指向某一种类型的存储单元，这种数据类型是在指针类型的定义中确定的，称为指针类型的基类型。指针类型定义如下：

类型名＝^基类型名；

例如：type q=^integer;

var a,b,c:q;

说明 q 是一指向整型存储单元的指针类型，其中"^"为指针符。a,b,c 均定义为指针变量，分别可以指向一个整型存储单元。

上例也可定义为：

var a,b,c:^integer;

指针也可以指向有结构的存储单元。

例如：type person=record

name:string[10];

sex:(male,female);

age:20..70

end;

var pt:^person;

pt 为指向记录类型 person 的指针变量。

2.动态变量

应用一个指针指向的动态存储单元即动态变量的形式如下：

指针变量名^

例如：p^、q^、r^

指针变量 p 和它所指向的动态变量^p 之间有如下关系：

P－>P'

以下语句把整数 5 存放到 p 所指向的动态变量 p^ 中去：

p^:=5;

以下语句把 p 所指向的 p^中的值赋给整型变量 i：

i:=p^;

如果指针变量 p 并未指向任何存储单元，则可用下列赋值语句：

p:=nil;

其中 nil 是 Turbo Pascal 保留字，表示“空”，相当于 C 里面的 null

- 对动态变量的操作

在 Turob Pascal 程序中，动态变量不能由 var 直接定义而是通过

调用标准过程 new 建立的。过程形式为：

new(指针变量名);

如果有下列变量定义语句：

var p:^integer;

仅仅说明了 p 是一个指向整型变量单元的指针变量，但这个整型单元并不存在，在指针变量 p 中还没有具体的地址值。在程序中必须通过过程调用语句：new(p);才在内存中分配了一个整型变量单元，并把这个单元的地址放在变量 p 中，一个指针变量只能存放一个地址。在同一时间内一个指针只能指向一个变量单元。当程序再次执行 new(p)时，又在内存中新建立了一个整型变量单元，并把新单元的地址存放在 p 中，从而丢失了旧的变量单元的地址。

为了节省内存空间，对于一些已经不使用的现有动态变量，应该使用标准过程 dispose 予以释放。过程形式为：dispose(指针变量名)；为 new(指针变量名)的逆过程，其作用是释放由指针变量所指向的动态变量的存储单元。例如在用了 new(p)后在调用 dispose(p)，则指针 p 所指向的动态变量被撤销，内存空间还给系统，这时

p 的值为 nil。

例：输入两个数，要求先打印大数后打印小数的方式输出，用动态变量做。

```
program dongtai;
  type intepter=^integer;
  var p1,p2:intepter;
```

```
  procedure swap(var,q1,q2:intepter);
       var p:integer;
  begin
       p:=q1;q1:=q2;q2:=p;
    end;
begin
  new(p1);new(p2);
    writeln('input 2 data: ');readln(p1^,p2^);
  if p1^        writeln('output 2 data: ',p1^:4,p2^:$);
end.
```

第二十二讲:数据结构

数据（Data）

数据是信息的载体。它能够被计算机识别、存储和加工处理，是计算机程序加工的"原料"。

随着计算机应用领域的扩大，数据的范畴包括：

整数、实数、字符串、图像和声音等。

数据元素（Data Element）

数据元素是数据的基本单位。数据元素也称元素、结点、顶点、记录。

一个数据元素可以由若干个**数据项**（也可称为字段、域、属性）组成。

数据项是具有独立含义的最小标识单位。

数据结构（Data Structure）

数据结构指的是数据之间的相互关系，即数据的组织形式。

1．数据结构一般包括以下三方面内容：

① 数据元素之间的逻辑关系，也称**数据的逻辑结构**（Logical Structure）；

数据的逻辑结构是从逻辑关系上描述数据，与数据的存储无关，是独立于计算机的。数据的逻辑结构可以看作是从具体问题抽象出来的数学模型。

② 数据元素及其关系在计算机存储器内的表示，称为**数据的存储结构**（Storage Structure）；

数据的存储结构是逻辑结构用计算机语言的实现（亦称为映象），

它依赖于计算机语言。对机器语言而言，存储结构是具体的。一般，只在高级语言的层次上讨论存储结构。

③ **数据的运算**，即对数据施加的操作。

数据的运算定义在数据的逻辑结构上，每种逻辑结构都有一个运算的集合。最常用的检索、插入、删除、更新、排序等运算实际上只是在抽象的数据上所施加的一系列抽象的操作。

所谓**抽象的操作**，是指我们只知道这些操作是"做什么"，而无须考虑"如何做"。只有确定了存储结构之后，才考虑如何具体实现这些运算。

为了增加对数据结构的感性认识，下面举例来说明有关数据结构的概念。

【例 1．1】 学生成绩表，见下表。

学生成绩表

学号	姓名	数学分析	普通物理	高等代数	平均成绩
880001	丁一	90	85	95	90
880002	马二	80	85	90	85
880003	张三	95	91	99	95
880004	李四	70	84	86	80
880005	王五	91	84	92	89
⋮	⋮	⋮	⋮	⋮	⋮

注意：在表中指出数据元素、数据项、开始结点和终端结点等概念

（1）逻辑结构

表中的每一行是一个数据元素（或记录、结点），它由学号、姓名、各科成绩及平均成绩等数据项组成。

表中数据元素之间的逻辑关系是：对表中任一个结点，与它相邻且在它前面的结点（亦称为直接前趋（Immediate Predecessor））最多只有一个；与表中任一结点相邻且在其后的结点（亦称为直接后继（Immediate Successor））也最多只有一个。表中只有第一个结点没有直接前趋，故称为开始结点；也只有最后一个结点没有直接后继。故称之为终端结点。例如，表中"马二"所在结点的直接前趋结点和直接后继结点分别是"丁一"和"张三"所在的结点，上述结点间的关系构成了这张学生成绩表的逻辑结构。

（2）存储结构

该表的存储结构是指用计算机语言如何表示结点之间的这种关系，即表中的结点是顺序邻接地存储在一片连续的单元之中，还是用指针将这些结点链接在一起？

（3）数据的运算

在上面的学生成绩表中，可能要经常查看某一学生的成绩；当学生退学时要删除相应的结点；进来新学生时要增加结点。究竟如何

进行查找、删除、插入，这就是数据的运算问题。

搞清楚了上述三个问题，也就弄清了学生成绩表这个数据结构。

2．数据的逻辑结构分类

在不产生混淆的前提下，常将数据的逻辑结构简称为数据结构。数据的逻辑结构有两大类：

（1）线性结构

线性结构的逻辑特征是：若结构是非空集，则有且仅有一个开始结点和一个终端结点，并且所有结点都最多只有一个直接前趋和一个直接后继。

线性表是一个典型的线性结构。栈、队列、串等都是线性结构。

（2）非线性结构

非线性结构的逻辑特征是：一个结点可能有多个直接前趋和直接后继。数组、广义表、树和图等数据结构都是非线性结构。

第二十三讲:数据结构之“串”

一、串的概念

串又称为字符串，是由 0 个或多个字符组成的有限序列。长度为 0 的串称为空串，它不包含任何字符。

串用'和'括起来。

二、串的运算

1.串的定义：

一般用一维数组实现串的运算，由此串的定义也用数组的形式来实现：

```
type
  stringtype=packed array[1..80] of char;
var
  s:stringtype;
```

另外，还有一种更简便的定义方法，利用 turbo pascal 中的 string 类型：

```
var
  s:string;
```

但是 string 类型有一个限制：运用 string 类型定义的数据长度只能是 1——255，也就是说不能超过 255 个字符。

2.串的标准函数

在 turbo pascal 中有如下标准函数可实现串的运算：

copy(s,x,y)：获取从 s 的第 x 个位置开始的 y 个字符

concat(s1,s2,...,sn)：相等于 s1+s2+...+sn

delete(s,x,y)：将 s 中从第 x 个位置开始的 y 个字符删去

insert(s1,s,x)：将 s1 插到 s 中的第 x 个位置

length(s)：获取 s 的长度

3.串的基本运算

(1)赋值

(2)连接

(3)求串长

(4)取子串

(5)求子串序号

(6)插入

(7)删除

(8)置换

三、串的匹配算法

示例：

四、练习题：

1.读入一英文句子，单词之间用空格或逗号隔开，统计其中单词个数，并输出各个字母出现的频率。(句子末尾不一定用"."结束)

2.一个句子，只含英文字母，单词间用空格或逗号作为分隔符。统计句子中的单词数，如果含有其他的字符，则只要求输出错误信息及错误类型。

含有大写字母 错误类型 error 1

数字（0-9) 错误类型 error 2

其他非法字符 错误类型 error 3

如 输入： It is 12!

输出： error 1 2 3

输入： i am ,a student

输出： 4

3.编码解码：从键盘输入一个英文句子，设计一个编码、解码程序。

编码过程：先键入一个正整数 N(1<=N<=26)。这个 N 决定了转换关系。 例如当 N=1，输入的句子为 ABCXYZ 时，则其转换码为 ABCXYZ 不变。当 N=2 时，其转换码为 BCDYZA，其它的非字母字符不变。为使编码较于破译，将转换码的信息自左而右两两交换，若最后仅剩单个字符则不换。然后，将一开始表示转换关系的 N 根据 ascii 表序号化成大写字母放在最前面。

如：abcABCxyzXYZ-/,1. n=3

① cdeCDEzabZAB-/,1. {根据 N 的值转换}

② dcCeEDazZbBA/-1,. {两两交换}

③ CdcCeEDazZbBA/-1,. {最后编码}

解码过程为编码的逆过程。

相关评论

流云于 2005-5-12 8:21:50

context 统计单词数

【题目】读入一英文句子，单词之间用空格或逗号隔开，统计其中单词个数，并输出各个

字母出现的频率。(句子末尾不一定用"."结束)

【算法分析】要注意连续两个空格或逗号与空格连在一起时的误判断。

例如输入以下字符串:"abc_abc___abc,_abc___,_abc_,_." ('_' 代表空格)

【参考程序】

```
var  a:string;  i,t:byte;  wor:boolean;j:char;
    b:array['A'..'Z'] of 65..90;
begin
    fillchar(b,sizeof(b),0);
    writeln('input a string:');
    readln(a);      t:=1;                          {t:单词数}
    for i:=1 to length(a) do begin                    {逐个字符检测}
       if (ord(upcase(a[i])) in [65..90] )then begin   {如果是字母}
          wor:=true;                               {则为某一单词的开头}
          b[(upcase(a[i]))]:=b[(upcase(a[i]))]+1;     {相应字母的个数加 1}
       end;
       if wor and((a[i]=' ') or (a[i]=',')) then {如为","或空格,且之前为单词}
           begin inc(t);wor:=false;end;        {单词结束,单词数加 1}
    end;
    if wor=false then dec(t);         {判断行末多余的空格或","}
    writeln('words:',t);              {输出单词总数}
```

```
    for j:='A' TO 'Z' do if b[j]<>0 then writeln(j,' ',b[j],'  '); {各字母个数}
end.
```

第二十四讲:数据结构之"栈"

1.栈的特点：

栈是一种线性表，对于它所有的插入和删除都限制在表的同一端进行，这一端叫做栈的“顶”，另一端则叫做栈的“底”，其操作特点是“后进先出”。

2.栈的一般定义：

```
type
   stack=record
            data:array[1..m] of datatype;
            t:0..m
   end;
var
   s:stack;
```

3.栈的基本运算：

(1)栈的插入 push(s,x)：往栈 st 中推入一个值为 x 的项目；

若 t=m 则 print('overflow')

否则 t:=t+1;data[t]:=x;

(2)栈的弹出 pop(s)：从栈 st 中弹出一个项目；

若 t=0 则 print('underflow')

否则 t:=t-1;

(3)读栈顶元素 top(s,x)：把栈顶元素的值读到变量 x 中，栈保持不变；

若 t=0 则 print('error')

否则 x:=data[t];

(4)判栈是否为空 sempty(s)：这是一个布尔函数，当栈 st 中没有元素(即 t=0)时，称它为空栈，函数取真值，否则值为假。

若 t=0 则 sempty:=true

否则 sempty:=false;

4.栈的应用之一——计算表达式的值

(1)表达式的三种形式：

中缀表达式：运算符放在两个运算对象中间，如：(2+1)*3；

后缀表达式：不包含括号，运算符放在两个运算对象的后面，所有的计算按运算符出现的顺序，严格从左向右进行（不再考虑运算符的优先规则，如：2 1 + 3 *；

前缀表达式：同后缀表达式一样，不包含括号，运算符放在两个运算对象的前面，如：* + 2 1 3。

(2)表达式的计算：

由于后缀表达式中没有括号，不需判别优先级，计算严格从左向右进行，故计算一个后缀表达式要比计算机一个中缀表达式简单得

多。

将中缀表达式转换为后缀表达式的算法思想：

·当读到数字直接送至输出队列中

·当读到运算符 t 时，

a.将栈中所有优先级高于或等于 t 的运算符弹出，送到输出队列中；

b.t 进栈

·读到左括号时总是将它压入栈中

·读到右括号时，将靠近栈顶的第一个左括号上面的运算符全部依次弹出，送至输出队列后，再丢弃左括号。

运用后缀表达式进行计算的具体做法：

·建立一个栈 S

·从左到右读后缀表达式，读到数字就将它转换为数值压入栈 S 中，读到运算符则从栈中依次弹出两个数分别到 Y 和 X，然后以“X 运算符 Y”的形式计算机出结果，再压加栈 S 中

·如果后缀表达式未读完，就重复上面过程，最后输出栈顶的数值则为结束

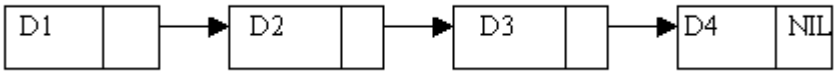
5.栈的应用之二——递归算法的非递归实现

第二十五讲:数据结构之"链表"

1 建立链表

链表是动态数据结构的一种基本形式。如果我们把指针所指的一个存贮单元叫做结点，那么链表就是把若干个结点链成了一串。我们把链表叫做动态数据结构，是因为链表中的结点可增可减，可多可少，可在中间任意一个位置插入和删除。

下面就是一个链表：



（图 T10．2）

每个链表有两个域，一个是数据域，一个是指向下一个结点的指针域。最后一个结点的指针域为 NIL，NIL 表示空指针。

要定义一个链表，每个结点要定义成记录型，而且其中有一个域为指针。

```
TYPE
  POINT=↑PP;
  PP=RECORD
    DATA：STRING[5];
    LINK：POINT;
    END;
VAR  P1，P2：POINT;
```

在这个定义中，定义了两个指针变量 P1，P2。每个结点有两个域，一个是数据域，一个是指针域，数据域是字符串型。这种定义中，指针中有记录，记录中有指针，形成一种递归调用。

下面我们来看一下如何定义上图的链表：

有：P1，P2，P3，P4 属于 POINT 类型。

（1）建立第一个结点

NEW（P1）；

P1↑.DATA:=D1;

(2) 建立第二个结点,并把它接在 P1 的后面

NEW(P2);

P2↑.DATA:=D2;

P1↑.LINK:=P2;

(3) 建立第三个结点,并把它接在 P2 的后面.

NEW(P3);

P3↑.DATA:=D3;

P2↑.LINK:=P3;

(4) 建立最后一个结点,并把它接在 P3 的后面.

NEW（P4）；

P4↑．DATA：＝D4；

P4↑．LINK：＝NIL；

P3↑．LINK：＝P4；

例 1　　建立一个有 10 个结点的链表，最后输出该链表。

P2 前一个结点，K 一直指向第一个结点。

```
PROGRAM  E1（INPUT，OUTPUT）；
  TYPE point=^pp;
    pp=record
      data:string[5];
      link:point;
      end;
  VAR
    p1,p2,k:point;
    i:integer;
  BEGIN
    {产生新结点 P1，作为链表的头}
    new(p1);
    writeln('input data');
    readln(p1^.data);
    {指针 K 指向链表头}
    k:=p1;
    {用循环产生 9 个新结点，每个结点都接在上一个结点之后}
    for i:=1 to 9 do
     begin
      new(p2);
```

```
        writeln('input data');
        readln(p2^.data);
        p1^.link:=p2;
        p1:=p2;
    end;
    {给最后一个结点的 LINK 域赋空值 NIL}
      p2^.link:=nil;
    {从链表头开始依次输出链表中的 DATA 域}
       while k^.link<>nil do
        begin
          write(k^.data,'->');
          k:=k^.link;
        end;
      END.
```

在本程序里，输出链表的过程就是一个链表的遍历。给出一个链表的头结点，依次输出后面每一个结点的内容，指针依次向后走的语句用 K：＝K↑．LINK 来实现。

例 2　　读入一批数据，遇负数时停止，将读入的正数组成先进先出的链表并输出。

分析：首先应定义指针类型，结点类型和指针变量，读

入第一个值，建立首结点，读入第二个值，判断它是否大于零，若是，建立新结点。

```
PROGRAM fifo(input,output);
{建立先进先出链表}
TYPE
 Point=^node;
 Node=RECORD
Data:real;
Link:point
END;
        VAR
         head,last,next:point;
         x:real;
        BEGIN
        {读入第一个值，建立首结点}
         read(x);
         write(x:6:1);
         new(head);
         head^.data:=x;
         last:=head;
         {读入第二个值}
         read(x);
```

```
        write(x:6:1);
        WHILE x>=0 DO
        BEGIN
        {建立新结点}
        new(next);
        next^.data:=x;  {链接到表尾}
        last^.link:=next; {指针下移}
        last:=next;      {读入下一个值}
        read(x);
        write(x:6:1)
        END;
       Writeln;          {表尾指针域置 NIL}
       Last^.link:=NIL;  {输出链表}
       Next:=head;
       WHILE next<>NIL DO
BEGIN
   Write(next^.data:6:1);
   Next:=next^.link
END;
  Writeln
       END.
```

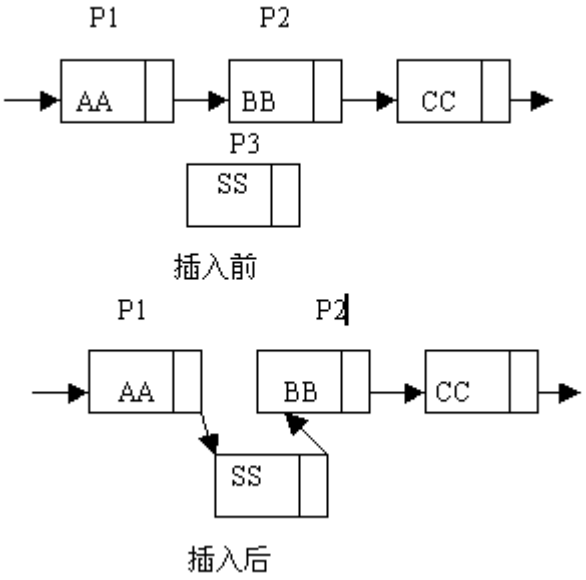
例 3　　读入一批数据，遇负数时停止，将读入的正数组成先进后出的链表并输出。

```
PROGRAM fifo(input,output);
{建立先进后出链表}
TYPE
 point=^node;
 node=RECORD
data:real;
link:point
END;
        VAR
         head,last,next:point;
         x:real;
        BEGIN
        {初始准备}
         next:=NIL;
         read(x);  {读入第一个数}
         write(x:6:1);
         WHILE x>=0 DO
       BEGIN {建立一个新结点}
       new(head);
       head^.data:=x;  {链接到表首}
```

```
        head^.link:=next;{指针前移}
        next:=head;      {读入下一个数}
        read(x);
        write(x:6:1)
END;
          writeln;        {输出链表}
          WHILE next<>NIL DO
BEGIN
  write(next^.data:6:1);
  next:=next^.link
END;
          writeln
       END.
```

2　在链表中插入结点

在一个已建好的链表中插入一个结点，这是一种常见算法。当我们找到插入点后，就断开原先的链接，将新结点插入进来。



（图 T10．3）

如图所求示：要把 P3 的结点插入到 P2 之前，应该这样操作：

（1）　　P1，P2 之间的链断开，改为 P1 指向 P3。

P1↑．LINK：＝P3；

（2）　　将 P2，P3 连接起来：

P3↑．LINK：＝P2；

于是，P3 所指向的结点便插入进来了。

例 4　　插入一结点的过程

```
  PROCEDURE insert(x:real;VAR head:point);
   {插入一结点的过程}
  VAR
q,last,next:point;
BEGIN
 {建立新结点}
```

```
new(q);
q^.data:=x;
IF X<=head^.ata
   THEN {插入前表}
        BEGIN
          q^.link:=head;
          head:=q;
        EDN
   ELSE BEGIN   {找出表中合适的位置}
        Next:=head;
        WHILE (x>next^.data) AND (next^.link<>NIL) DO
           BEGIN
            Last:=next;
            Next:=next^.link
           END;
        IF x<=next^.data
           THEN  {插入中间表}
               BEGIN
                 last^.link:=q;
                 q^.link:=next
               END
           ELSE   {插入表尾}
```

```
            BEGIN
              next^.link:=q;
              q^.link:=NIL
            END
        END
END;
```

例 5　　建立一个有序的整数链表，再将一任意整数插入进链表，使链表仍然有序。

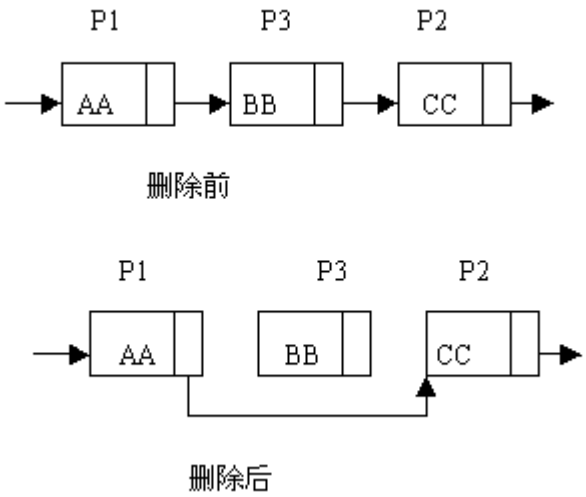
```
PROGRAM e1(input,output);
 TYPE point=^pp;
      pp=record
         data:integer;
         link:point;
         end;
 VAR
    p1,p2,p3,k:point;
  BEGIN
     {建立一个整数的链表，要求输入的整数依次是有序的，以数 9999 结束}
    new(p1);
    writeln('input data');
```

```
            readln(p1^.data);
            k:=p1;
            repeat
               new(p2);
               writeln('input data');
               readln(p2^.data);
               p1^.link:=p2;
               p1:=p2;
             until p2^.data=9999;
             p2^.link:=nil;{有序链表建立完毕}
            writeln('input p3');
            readln(p3^.data);
            p1:=k;p2:=k;{P1，P2 都指向表头}
             {P2 找到插入点后一个结点}
            while p2^.data<p3^.data do
p2:=p2^.link;
             {P1 找到插入点以前的结点}
            while p1^.link<>p2 do
              p1:=p1^.link;
             {将 P3 接入 P1，P2 之间}
             p1^.link:=p3;
            p3^.link:=p2;
```

```
                {将整个插入后的链表依次打印出来}
        repeat
                        write(k^.data,'->');
                        k:=k^.link;
                until k^.data=9999;
                write(k^.data);
            END.
```

3　　删除一个结点

要在一个链表中删去一个结点，首先在链表中找到该结点，将其前面的指针与该点断开，直接接到后面的结点，再用 DISPOSE　命令将 P3 结点空间释放。如图，删除 P3 结点：



（图 T10．4）

删除语句如下：

P1↑．LINK：＝P3↑．LINK

```
DISPOSE（P3）；
```

例 6　　删除结点的过程

```
PROCEDURE delete(x:real;VAR head;point;VAR deleted:Boolean);

{删除结点的过程}
VAR
  Last,next:point;
BEGIN
{遍历表直到找到目标或到达表末}
next:=head;
 WHILE(next^.data<>x)AND(next^.link<>DIL) DO
   BEGIN
    Last:=next;
    Next:next^.link
   END;
{如果目标文件找到，删除包含它的结点，设置 deleted}
  IF next^.data=x
THEN BEGIN
   Deleted:=true;
   IF next=head
   THEN head:=head^.link
   ELSE last^.link:=next^.link
```

```
 END
ELSE deleted:=false
END;
```

例 7　　在一个有序链表中装有 1 到 100 共 100 个整数。要求任意输入 100 以内的一个整数，把它从链表中删除。

```
PROGRAM e1(input,output);
 TYPE point=^pp;
      pp=record
         data:integer;
         link:point;
         end;
 VAR
    p1,p2,p3,k:point;
    i:integer;
 BEGIN
   {建立一个 1－－100 的整数的有序链表}
    new(p1);
    k:=p1;
    p1^.data:=1;
    for i:=2 to 100 do
      begin
```

```
                new(p2);
                p2^.data:=i;
                p1^.link:=p2;
                p1:=p2;
              end;
p2^.link:=nil;
{输入要删除的结点的 DATA 值}
          new(p3);
          writeln('input p3');
          readln(p3^.data);
          P1:=k;p2:=k;
           {P2 指针查找 P3 结点}
          while p2^.data<p3^.data do
p2:=p2^.link;
           {P1 指针定位于 P3 之前}
          while p1^.link<>p2 do
            p1:=p1^.link;
          p1^.link:=p2^.link;
          dispose(p3);{将 P3 结点的空间释放}
           {输出修改后的链表}
          repeat
               write(k^.data,'->');
```

```
        k:=k^.link;
    until k^.data=100;
    write(k^.data);
END.
```

第 26 讲:队列

1.队列的特点：

队列也是一种线性表，对于它所有的插入都在队列的一端进行，所有的删除都在另一端进行，进行删除的一端叫队列的“头”，进行插入的一端叫队列的“尾”，其操作特点是“先进先出”。

2.队列的一般定义：

```
type
  queue=record
        data:array[1..m] of datatype;
        head,tail:1..m
  end;
var
  q:queue;
```

3.队列的基本操作：

(1)队列的插入 enq(q,x)：在队列 q 中插入一个值为 x 的元素，也称为进队；

q[tail]:=x;

若 tail=m 则 tail:=1

否则 tail:=tail+1;

若 tail=head 则 print('overflow')

(2)队列的删除 deq(q)：从队列 q 中删除一个元素，也称为出队；

若 tail=head 则 print('underflow')

否则

若 head=m 则 head:=1 否则 head:=head+1;

(3)读队头元素 head(q,x)：将队列 q 中头部元素的值读到 x 中；

若 tail=head 则 print('error')

否则 x:=q[head];

(4)判队列是否为空 qempty(q)：这是一个布尔函数，当 q 是空队列时，取真值，否则取假值。

若 tail=head 则 qempty:=true

否则 qempty:=false;

3.队列的应用：

例：设有一个表，记为 L=(a1,a2,a3,...,an)，其中

L——表名

a1,a2,a3,..,an——表中元素。

当 ai 为数值时表示元素，当 ai 为大写字母时，表示另一个表，

但不能循环定义。例如，下列的定义是合法的（约定 L 是第一个表的表名）：

L=(3.4,3,4,K,8,0,8)

K=(15,5,8,P,9,4)

P=(4,7,8,9)

程序要求：当全部表给出后，示出所有元素的最大元素。

思路：

(1)可以建立两种队列，一种队列是存放数值的数值队列 Q1，并有 Q1[A]到 Q1[Z]共 26 条数值队列，另一种队列 Q2 存放字母，表示将要向该字母所对应的数值队列输入数值的事件队列。

(2)根据题目要求，事件队列 Q2 中应先放入字母 L，表示首先输入 L 数值队列的数据。

(3)从事件队列 Q2 中取出一个字母，并招待该字母对应的数值队列的数值输入，如果输入的数据中包含字母，则把字母放入事件队列中，并记录输入的数值中的最大值。

(4)重复(3)步骤，直到事件队列中为空。

第 27 讲:树

树(Tree)是 n(n>=0)个结点的有限集。在一棵非空树中：

(1) 有且仅有一个特定的称为根的结点;

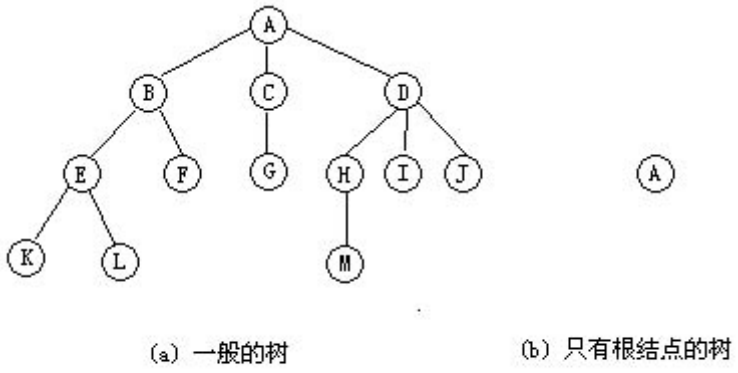
(2) 当 n>1 时其余结点可分为 m(m>0)个互不相交的有限集

T1,T2...Tm,其中,每一个集合

本身 又是一棵树, 并且称为根的子树(subtree)例如,在图

6.1 中,(a)是只有一个根

结点的树;(b)是有 13 个结点的树,其中 A 是根,其余结点分成三个互不相交的子树：



(a) 一般的树　　(b) 只有根结点的树

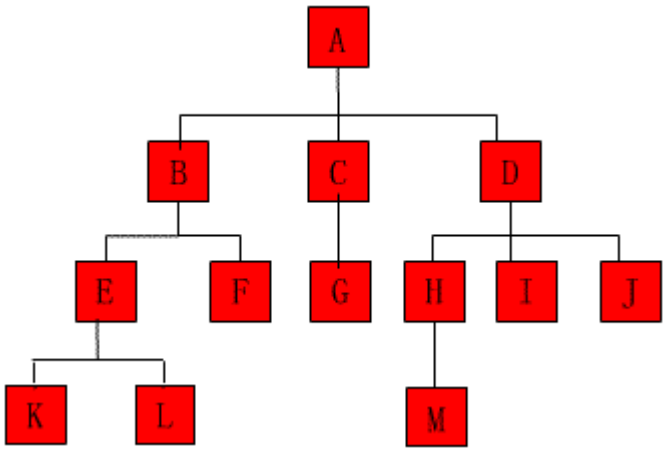
图6.1 树的示例

树是一种数据结构 : Tree=(D,R) 其中：D 是具有相同特性的数据元素的集合;若 D 只含一个数据元素,则 R 为空集,否则 R 是 D 上 个二元关系的集合，即 R={H}。H 为如下描述二元关系：

- (1) 在D中存在发唯一的称为根的数据元素,它在关系H下无前驱;
- (2) 若 D-{root}≠Φ，则存在 D-{root}的一个划分 D1,D2,...Dm(m>0),对任意一对 j≠k ??? (1<=j,k<=m)有 Dj∩Dk=Φ，且对任意的 i(1<=i<=m)，唯一存在数据元素 Xi∈Di，有 ∈H;

- (3) 对应于 D-{root}的划分，H-{ ,...， }有唯一的一个划分 H1,H2,...， Hm(m>0),对任意一对 j≠k(1<=j,K<=m)有，Hj∩Hk=φ，且对任意的 i(1<=i<=m)Hi 是 Di 上的二元关系，(Di,{Hi})是一棵符合本定义的树，称为根 root 的子树。

一、树的基本术语



1.树的度——也即是宽度，简单地说，就是结点的分支数。以组成该树各结点中最大的度作为该树的度，如上图的树，其度为 3;

2.树的深度——组成该树各结点的最大层次，如上图，其深度为 4;

3.森林——指若干棵互不相交的树的集合，如上图，去掉根结点 A，其原来的二棵子树 T1、T2、T3 的集合{T1,T2,T3}就为森林;

4.有序树——指树中同层结点从左到右有次序排列，它们之间的次序不能互换，这样的树称为有序树，否则称为无序树。

- 结点的度：结点拥有的子树数。

- 叶子（终端结点）：度为零的结点。
- 非终端结点（分支结点）：度不为零的结点。
- 树的度：树内各结点的度的最大值。

二、树的表示

树的表示方法有许多，常用的方法是用括号：先将根结点放入一对圆括号中，然后把它的子树由左至右的顺序放入括号中，而对子树也采用同样的方法处理；同层子树与它的根结点用圆括号括起来，同层子树之间用逗号隔开，最后用闭括号括起来。如上图可写成如下形式：

(A(B(E(K,L),F),C(G),D(H(M),I,J)))

三、树的基本操作：

(1) INITIATE(T) 初始化操作，置 T 为空树。

(2) ROOT(T)\ROOT(X) 求根函数。求数 T 的根或求结点 x 所在的树的根结点。若 T 是空树或 X 不在任何一棵树上，则函数值为“空”。

(3) RARENT(T,x) 求双亲函数。求树 T 中结点 x 的双亲结点。若结点 x 是树 T 的根结点或结点 x 不在 T 中，则函数值为“空”。

(4) CHILD(T,x,i) 求孩子结点函数。求数 T 中结点 x 的第 i 个孩子结点。若结点 x 是树 T 的叶子或无第 i 个孩子或结点 x 不在树 T 中，则函数值为“空”。

(5) RIGHT_SINLING(T,x)求右兄弟函数。求树T中结点x右边的兄弟。若结点x是其双亲的最右边的孩子结点或结点x不在树T中，则函树值为“空”。

(6) CRT_TREE(x,F) 建树操作。生成一棵以X结点为根，以森F为子树森林的树。

(7) INS_CHILD(y,i,x) 插入子树操作。置以结点x为根的树为结点y的第i棵子树。若原树中无结点y或结点y的子树个数&gti-1，则空操作。

(8) DEL_CHILD(x,i) 删除子树操作。删除结点x的第i棵子树。若无结点x或结点x的子树个数&gti，则空操作。

(9) TRAVERSE(T) 遍历操作。按某个次序依此访问树中各个结点，并使每个结点只被访问一次。

(10) CLEAR(T) 清除结构操作。将树T置为空树。

四、

5.2.1 二叉树定义与基本操作

二叉树 (binary tree) 是另一种树型结构，它的特点是每个结点至多只有二棵子 树 (即二叉树中不存在度大于 2 的结点)，并且，二叉树的子树有左右之分，其次序不能任意颠倒 . 二叉树是一种数据结构 :

Binary_tree=(D,R)

其中： D 是具有相同特性的数据元素的集合 ;若 D 等于空 ,则 R 等于空称为空的二叉树 ;若 D 等于空则 R 是 D 上某个二元关系 H 的集合，即 R={H}，且

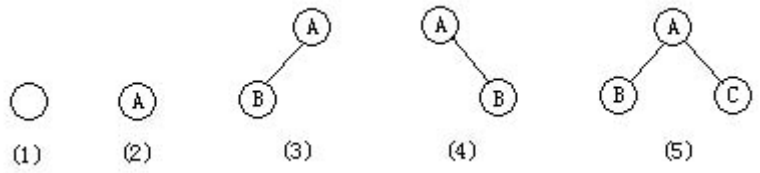
(1) D 中存在唯一的称为根的元素 r，它的关系 H 下无前驱 ;

(2) 若 D-{r}不等于空，则 D-{r}={Dl,Dr}，且 Dl 交 Dr 等于空 ;

(3) 若 Dl 不等于空 ,则在 Dl 中存在唯一的元素 xl,〈 r,xl〉属于 H，且存在 Dl 上的关系 Hl 属于 H； 若 Dr 不等于空 ,则在 Dr 中存在唯一的元素 xr,〈 r,xr〉 >属于 H, 且存 Dr 上的关 系 Hr 属于 H； H={r,xl,＜ r,xr＞ ,Hl, Hr};

(4) (Dl, Hl) 是一棵合本定义的二叉树，称为根 r 的左子树 ,(Dr,Hr) 是一棵符合定义的二叉树，称为根的右子树。

其中，图 6.2 是各种形态的二叉树 .



(1) 为空二叉树 (2)只有一个根结点的二叉树 (3)右子树为空的二叉树 (4)左子树为空的二叉树 (5)完全二叉树

二叉树的基本操作：

(1)INITIATE(BT ） 初始化操作。置 BT 为空树。

(2)ROOT(BT)\ROOT(x) 求根函数。求二叉树 BT 的根结点或求结点 x 所在二叉树的根结点。

若 BT 是空树或 x 不在任何二叉树上，则函数值为 “空 ”。

(3)PARENT(BT,x) 求双亲函数。求二叉树 BT 中结点 x 的双亲结点。若结点 x 是二叉树 BT 的根结点

或二叉树 BT 中无 x 结点，则函数值为 “空 ”。

(4)LCHILD(BT,x) 和 RCHILD(BT,x) 求孩子结点函数。分别求二叉树 BT 中结点 x 的左孩 子和右孩子结点。

若结点 x 为叶子结点或不在二叉树 BT 中，则函数值为 “空 ”。

(5)LSIBLING(BT,x) 和 RSIBING(BT,x) 求兄弟函数。分别求二叉树 BT 中结点 x 的左兄弟和右兄弟结点。

若结点 x 是根结点或不在 BT 中或是其双亲的左 /右子树根 ，则函树值 为 “空 ”。

(6)CRT_BT(x,LBT,RBT) 建树操作。生成一棵以结点 x 为根，二叉树 LBT 和 RBT 分别为左， 右子树的二叉树。

(7)INS_LCHILD(BT,y,x) 和 INS_RCHILD(BT,x) 插入子树操作。将以结点 x 为根且右子树为空的二叉树

分别置为二叉树 BT 中结点 y 的左子树和右子树。若结点 y 有左子树 /右子树，则插入后是结点 x 的右子树。

(8)DEL_LCHILD(BT,x) 和 DEL-RCHILD(BT,x) 删除子树操作。分别删除二叉树 BT 中以结点 x 为根的左子树或右子树。

若 x 无左子树或右子树，则空操作。

(9)TRAVERSE(BT) 遍历操作。按某个次序依此访问二叉树中各个结点，并使每个结点只被访问一次。

(10)CLEAR(BT) 清除结构操作。将二叉树 BT 置为空树。

5.2.2 二叉树的存储结构

一 、顺序存储结构

连续的存储单元存储二叉树的数据元素。例如图 6.4(b)的完全二叉树 , 可以向量 (一维数组) bt(1:6)作它的存储结构，将二叉树中编号为 i 的结点的数据元素存放在分量 bt[i]中 ,如图 6.6(a) 所示。但这种顺序存储结构仅适合于完全二叉树 ,而一般二叉树也按这种形式来存储 ,这将造成存 贮浪费。如和图 6.4(c)的二叉树相应的存储结构图 6.6(b)所示，图中以 “0”表示不存在此结点 .

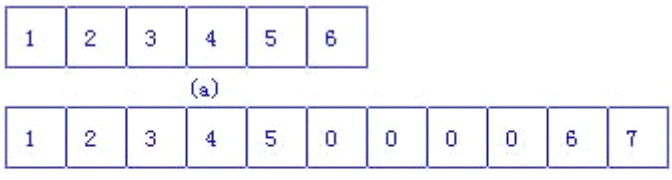


图6.6二叉的顺序存储结构
(a)完全二叉树；(b)一般二叉树

二、 链式存储结构

由二叉树的定义得知二叉树的结点由一个数据元素和分别指向左右子树的两个分支构成 ,则表 示二叉树的链表中的结点至少包含三个域 :数据域和左右指针域 ,如图 (b)所示。有时 ,为了便于找 到结点的双亲 ,则还可在结点结构中增加一个指向其双亲受的指针域，如图 6.7(c)所示。

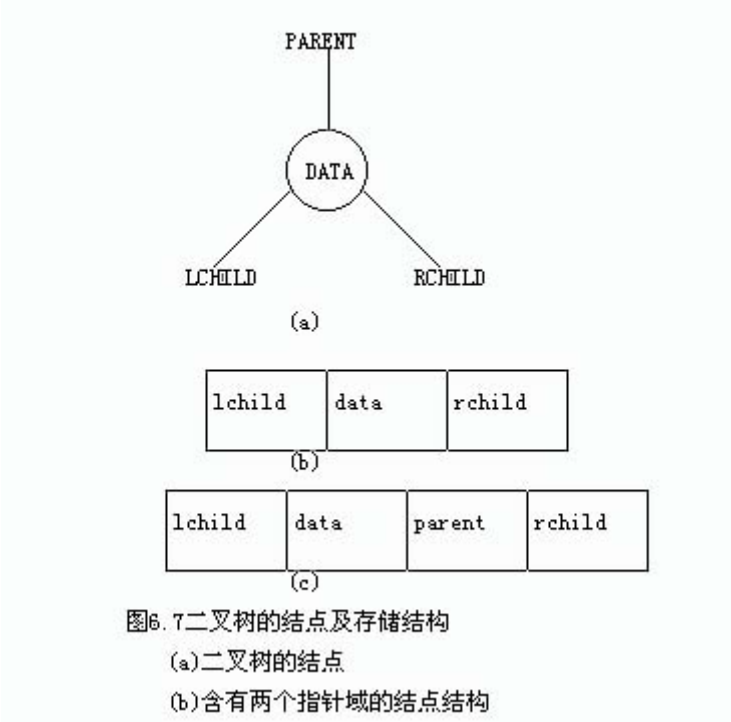


图6.7二叉树的结点及存储结构

(a)二叉树的结点

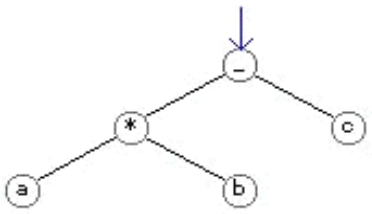
(b)含有两个指针域的结点结构

5.3 遍历二叉树

遍历二叉树 (traversing binary tree)的问题， 即如何按某条搜索路径巡访树中每个结点，使得每个结点均被访问一次，而且仅被访问一次。其中常见的有三种情况：分别称之为先 (根)序遍历，中 (根)序遍历和后 (根)序遍历。

5.3.1 前序遍历

前序遍历运算：即先访问根结点，再前序遍历左子树，最后再前序遍历右子树。前序遍历运算访问二叉树各结点是以根、左、右的顺序进行访问的。例如：



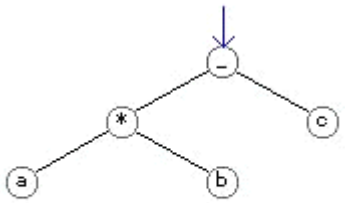
一棵简单的二叉树

按前序遍历此二叉树的结果为： Hello!How are you?

```
proc preorder(bt:bitreprtr)
  if (bt<>null)[
    print(bt^);
    preorder(bt^.lchild);
    preorder(bt^.rchild);]
end;
```

5.3.2 中序遍历

中序遍历运算：即先中前序遍历左子树，然后再访问根结点，最后再中序遍历右子树。中序遍历运算访问二叉树各结点是以左、根、右的顺序进行访问的。例如：



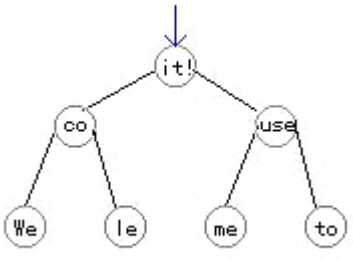
一棵简单的二叉树

按中序遍历此二叉树的结果为： a*b-c

```
proc inorder(bt:bitreprtr)
  if (bt<>null)[
    inorder(bt^.lchild);
    print(bt^);
    niorder(bt^.rchild);]
end;
```

5.3.3 后序遍历

后序遍历运算：即先后序遍历左子树，然后再后序遍历右子树，最后访问根结点。后序遍历运算访问二叉树各结点是以左、右、根的顺序进行访问的。例如：



一棵简单的二叉树

按后序遍历此二叉树的结果为： Welecome to use it!

```
proc postorder(bt:bitreprtr)
  if (bt<>null)[
    postorder(bt^.lchild);
    postorder(bt^.rchild);]
    print(bt^);
end;
```

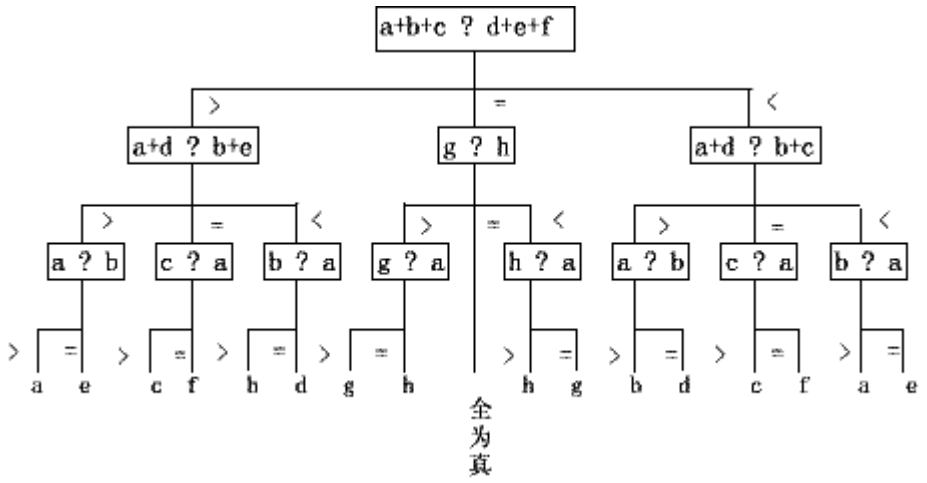
五、例：

1.用顺序存储方式建立一棵有 31 个结点的满二叉树，并对其进行先序遍历。

2.用链表存储方式建立一棵如图三、4 所示的二叉树，并对其进行先序遍历。

3.给出一组数据：R={10.18,3,8,12,2,7,3}，试编程序，先构造一棵二叉树，然后以中序遍历访问所得到的二叉树，并输出遍历结果。

4.给出八枚金币 a,b,c,d,e,f,g,h，编程以称最少的次数，判定它们蹭是否有假币，如果有，请找出这枚假币，并判定这枚假币是重了还是轻了。



第 28 讲:图

一、图的概念

1 ．图的定义

图 1.3-1 所示的 ⑴， ⑵， ⑶均为图 (Graph)，它有若干个不同的点 $v1$， $v2$， …， vn，在其中一些点之间用直线或曲线连接。图中的这些点被称为顶点 (vertex)或结点，连接顶点的曲线或直线称为边 (edge)。通常将这种由若干个顶点以及连接某些顶点的边所组成的图形称为图，顶点通常被称作是图中的数据元素。

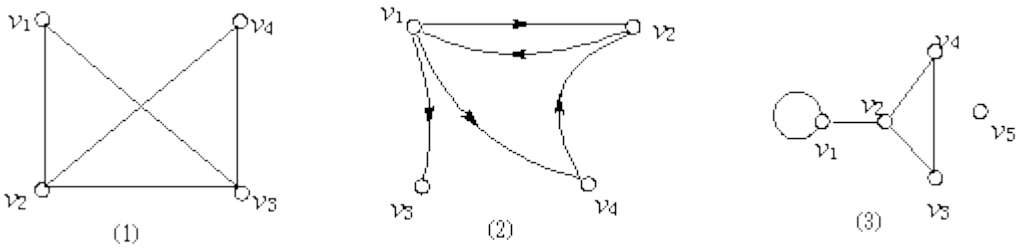


图 1.3-1

在线性结构中每个元素只有一个前趋和一个后续，而图 1.3-1 中的各个图则与之不同，它是一种较为复杂的非线性数据结构，在图结构中的任意两个元素之间都可能相互联系，即每个元素都可能有多个前趋或多个后续。图作为一种数据结构，通常又可被定义为： graph=(V，E)或 G=(V， E)，即一个图是由顶点的集合 V 和边的集合 E 组成。

在图 1.3-1 中 ⑴图中的边没有方向，这类图称为无向图 (undirected graph)。在记录无向图时， (*v* 1， *v* 2)等价于 (*v* 2， *v* 1)。

在图 1.3-1 中 ⑵图中的边上有一个箭头，它表示边的方向，这类图称为有向图 (directed graph)。在记录有向图时， <*v* 1， *v* 2 >与 <*v* 2，*v* 1 >是两条不同的边。

图 1.3-1 中 ⑴图的顶点集合为：

V ＝｛ *v* 1， *v* 2， *v* 3， *v* 4｝

边集合为：

E ＝{（ $v1$， $v2$），（ $v1$， $v3$），（ $v2$， $v3$），（ $v2$， $v4$），（ $v3$， $v4$）}

图 1.3-1 中 ⑵图的顶点集合为：

V ＝{ $v1$， $v2$， $v3$， $v4$}

边集合为：

E ＝{ <$v1$， $v2$>， <$v1$， $v3$>， <$v1$， $v4$>， <$v2$， $v1$>， <$v4$， $v2$>}

2 ．图的常用术语

环 (cycle)：图 1.3-1 中 ⑶图中的 $v1$ 点本身也有边相连，这种边称为环。

有限图：顶点与边数均为有限的图，如图 1.3-1 中的三个图均属于有限图。

简单图：没有环且每两个顶点间最多只有一条边相连的图，如图 1.3-1 中的 ⑴图。

邻接与关联：当（ $v1$， $v2$） ∈E，或 <$v1$， $v2$>∈E，即 $v1$， $v2$ 间有边相连时，则称 $v1$ 和 $v2$ 是相邻的，它们互为邻接点（ adjacent），同时称（ $v1$， $v2$）或 <$v1$， $v2$>是与顶点 $v1$、$v2$ 相关联的边。

顶点的度数 (degree)：从该顶点引出的边的条数，即与该顶点相关联的边的数目，简称度。图 1.3-1 中 ⑴、⑵图的各顶点的度见下表：

顶点	*v* 1	*v* 2	*v* 3	*v* 4
⑴ 图	2	3	3	2
⑵ 图	4	3	1	2

入度（ indegree）：有向图中把以顶点 *v* 为终点的边的条数称为是顶点 *v* 的入度。

出度（ outdegree）：有向图中把以顶点 *v* 为起点的边的条数称为是顶点 *v* 的出度。图 1.3-1 中、⑵图各顶点的入度和出度见下表 (各顶点的入度与出度之和为该顶点的度)：

顶点	*v* 1	*v* 2	*v* 3	*v* 4
入度	1	2	1	1
出度	3	1	0	1

终端顶点：有向图中把出度为 0 的顶点称为终端顶点，如图 4.3-1 中 ⑵图的 *v* 3。

道路与路长：道路也称路径 (path)。在图 G=(V， E)中，如果存在由不同的边 (*v* i0， *v* i1)， (*v* i1， *v* i2)， …， (*v* in-1， *v* in)或是 <*v* i0， *v* i1 >， <*v*i 1， *v* i 2>， …， < *v* in-1， *v* in >)组成的序列，则称顶点 *v* i0， *v* in 是连通的，顶点序列（ *v* i0， *v* i1， *v* i2， …，

v in）是从顶点 *v* i0 到顶点 *v* in 的一条道路。路长是道路上边的数目，*v* i0 到 *v* in 的这条道路上的路长为 n。

连通图：对于图中任意两个顶点 *v* i、 *v* j ∈V， *v* i、 *v* j 之间有道路相连，则称该图为连通图 (connected graph)，如 1.3-1 中的 ⑴图。

带权图：给图 1.3-1 中各图的边上附加一个代表性数据 (比如表示长度、流量或其他)，则称其为带权图，如图 1.3-2。

网络：带权的连通图，如图 1.3-2 所示。

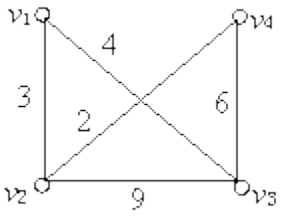


图 1.3-2

二、图的存储

图的最常见的存储方式是用邻接矩阵和邻接表。

1. 邻接矩阵存储

⑴ 邻接矩阵

邻接矩阵 (Adjacency Matrix)是表示顶点间邻接关系的矩阵。在图的邻接矩阵表示法中通常用一个邻接矩阵表示顶点间的相邻关系，另外用一个顺序表来存储顶点信息。

具有 n 个顶点的图 G=(V， E)的邻接矩阵可以定义为：

$$A[i,j]=\begin{cases}1, \text{如果}(v_i, v_j)\text{或}\langle v_i, v_j\rangle \in E \\ 0, \text{如果}(v_i, v_j)\text{或}\langle v_i, v_j\rangle \in E\end{cases}$$

图 1.3-1 中 ⑴图和 ⑵图的邻接矩阵表示为：

	v_1	v_2	v_3	v_4
v_1	0	1	1	0
v_2	1	0	1	1
v_3	1	1	0	1
v_4	0	1	1	0

	v_1	v_2	v_3	v_4
v_1	0	1	1	1
v_2	1	0	0	0
v_3	0	0	0	0
v_4	0	1	0	0

带权图的邻接矩阵可以定义为：

$$A[i,j]=\begin{cases}W_{ij}, \text{如果}(v_i, v_j)\text{或}\langle v_i, v_j\rangle \in E \\ \infty, \text{如果}(v_i, v_j)\text{或}\langle v_i, v_j\rangle \in E\end{cases}$$

图 1.3-2 邻接矩阵表示为：

	v_1	v_2	v_3	v_4
v_1	∞	3	4	∞
v_2	3	∞	9	2
v_3	4	9	∞	6
v_4	∞	2	6	∞

⑵ 建立已知图的邻接矩阵

如要建立图 1.3-1 中 ⑴、⑵的已知图，则可用常量数组直接说明如下：

```
const   graph1:array[1..4,1..4] of
integer=((0,1,1,0),(1,0,1,1),(1,1,0,1),(0,1,1,0));

graph2:array[1..4,1..4] of integer=((0,1,1,1),(1,0,0,0),(0,0,0,0),(0,1,0,0));
```

同样图 1.3-2 中的图也可用常量数组直接说明如下：

```
const   graph3:array[1..4,1..4] of
integer=((0,3,4,0),(3,0,9,2),(4,9,0,6),(0,2,6,0));
```

⑶ 建立任意带权无向图的邻接矩阵

程序如下：

```
Program   adjmatrix(input,output);
var   vi,vj,vn,ei,en,wn:integer;
```

```
graph:array[1..20,1..20] of integer;
Begin
    read(vn,en);  { 读入顶点数和边数 }
for vi:=1 to vn do
      for vj:=1 to vn do
           graph[vi,vj]:=0;
for ei:=1 to en do
begin
      read(vi,vj,wn);  { 读入每条边的两个顶点及边上的权值 }
      graph[vi,vj]:=wn;
      graph[vj,vi]:=wn
end;
for vi:=1 to vn do
begin
      for vj:=1 to vn do write(graph[vi,vj]:8);  { 输出图 }
      writeln
end
End.
```

2. 邻接表存储

⑴ 邻接表

邻接矩阵采用的是顺序存储的方式，而邻接表 (Adjacency List)采用的是图的一种链式存储方式。邻接表包含了一个顶点顺序表和每个顶点对应的单链表。

顶点顺序表中的顶点与图中的顶点一一对应，每个顶点的单链表中的每个顶点包含了邻接点域（用来指示与该顶点邻接的点在图中的位置），链域（用来指示相邻的另一条边的顶点），数据域（用来存储边的权值等信息）。

三、图的遍历

与树的遍历类似，从图中的某一顶点出发有序访问图中其余的所有顶点，并使每一个顶点恰好被访问一次，这个过程称为是图的遍历（ traversing graph）。在进行图的遍历时，由于图中顶点间是多对多的关系，图中的任一顶点都可能和其余的顶点相邻接。为避免重复访问，在遍历图的过程中，必须对访问过的顶点作上标记。如设置一个辅助的布尔型数组 visited[v1..vn]，将该数组的初始值设为假，一旦顶点 vi 被访问，便将其值 visited[vi]设为真。

常用的图的遍历方法有两种：深度优先遍历和广度优先遍历，它们对有向图和无向图均适用。

1. 深度优先遍历

深度优先遍历 (Depth-first Traversal)同树的先序遍历比较类似。先假设图中的所有顶点均未被访问，遍历时从图中的某个顶点 v0 出发，访问该顶点，然后依次从 v0 的未被访问的邻接点出发深度优先遍历图，直到图中所有和 v0 有路径相通的顶点都被访问到。如果这时图中还有顶点未被访问，则另选图中一个未被访问的顶点作起始点，重复上述过程，直到图中所有顶点均被访问到为止。

根据深度优先遍历的描述我们可以得到图 1.3-4 中的 ⑴图遍历的顶点序列为 V1→V2→V4→V8→V9→V5→V3→V6→V7，⑵图遍历的顶点序列为 V1→V2→V5→V3→V4→V6。

连通图的深度优先遍历的算法描述如下：（如果不是连通图则需多次使用遍历算法）

```
procedure dfs(vi:integer);
begin
    write(graph[vi].v:4);   {输出顶点数值}
    visited[vi]:=true;      { 设置已访问标记 }
    last:=graph[vi].link;   { 取边表指针 }
    while last<>nil do
    begin
        if not visited[last^.adjv] then dfs(last^.adjv);
        last:=last^.next
```

```
    end
end;
```

2. 广度优先遍历

广度优先遍历（ Breadth-first Traversal）也要先假设图中的所有顶点均未被访问，遍历时从某个顶点出发，访问该顶点后再依次访问与该顶点邻接的未被访问过的所有顶点。然后再分别从这些顶点出发进行广度优先遍历，直到图中所有未被访问过的顶点的相邻顶点均被访问到。如果这时图中仍有顶点为未被访问，则另选图中一个未被访问的顶点作为起点，再重复上述过程，直到图中所有顶点都被访问到为止。

根据广度优先遍历的描述我们可以得到图 1.3-4 中的 ⑴图遍历的顶点序列为 V1→V2→V3→V4→V5→V6→V7→V8→V9，⑵图遍历的顶点序列为 V1→V2→V3→V4→V5→V6。

在广度优先搜索过程中，设 Vi 是将被访问的未访问过的顶点。它的邻接点记为 Vi1， Vi2， …， Vin。访问了 Vi 后，将对应的 visit[Vi] 值设为真，并将 Vi 保存在队列中。当 Vi 出队时，搜索其邻接的 Vi1，Vi2， …， Vin 顶点，然后将其中未被访问的顶点者作上访问标记后放入队列中。这种方法是将每个已被访问过的顶点入队，保证了每个顶点最多只有一次入队。

四、图的应用

1 ．一笔画问题

数学家欧拉曾经解决过著名的七桥问题（七桥图见图 1.3-5 ⑴图）。下面写出七桥问题的描述：城市中有一条河，河中有 A、 D 两个岛，河上有七座桥来连接两个岛及河的 B、 C 两岸，问：⑴能否刚好经过每座桥一次，既无重复也无遗漏？ ⑵能否经过桥一次后又回到原来出发点上来？

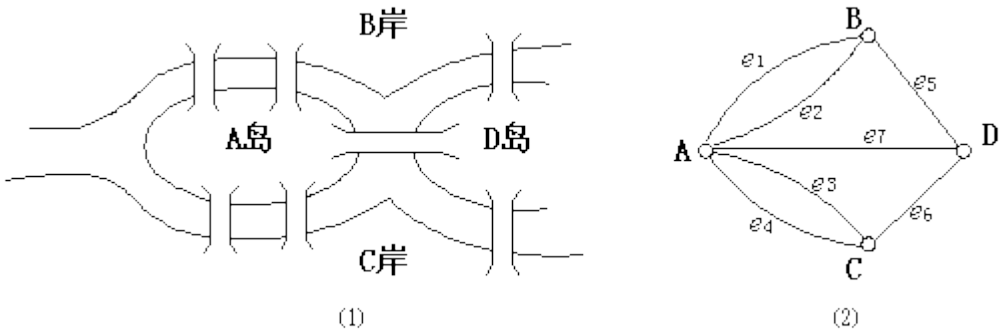


图 1.3-5

七桥问题可以画成图 1.3-5 中的 ⑵图的形式，这样七桥问题的第一问就转化成了能否一笔画成一个图的问题。

一个图能否一笔画成需要满足以下条件：先根据图的邻接矩阵求出每个顶点的度数。如果没有度数为奇数的顶点，则可以从任一点开始一笔画成一个图。如果有两个度数为奇数的顶点，则可从这两个奇数顶点中的任一点开始一笔画成一个图。如果度数为奇数的顶点超过两个，则这个图不能够一笔画出。

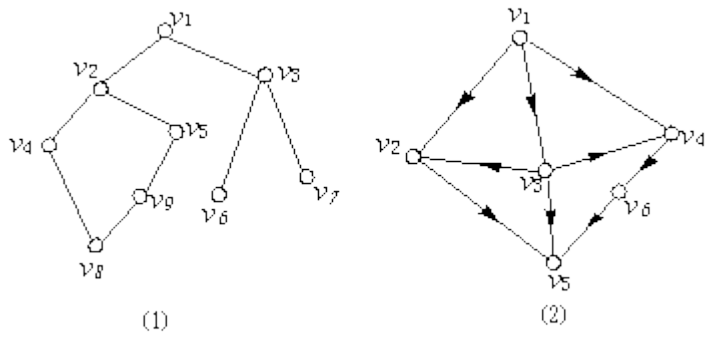


图 1.3-6

对于图 1.3-5 的 ⑵图或是 1.3-6 所示的无向图，可以用数组 graph 存储图的邻接矩阵，用数组 degree 存储每个顶点的度数，用变量 Total_d 存储总的度数，用变量 Odd_num 存储度数为奇数的顶点个数，用变量 start 存储一笔画的起始顶点。

一笔画程序如下：

```
program stroke(input,output);
var graph:array[1..20,1..20] of 0..1;
    degree:array[1..20] of integer;
    odd_num,vn,vi,vj,start,total_d:integer;
begin
     odd_num:=0;total_d:=0;start:=1;
     write('please input the number of vertex:');
     readln(vn);
     writeln('please input the data:');
     for vi:=1 to vn do
```

```
    begin
        degree[vi]:=0;
        for vj:=1 to vn do
        begin
            read(graph[vi,vj]); {读入邻接矩阵}
            degree[vi]:=degree[vi]+graph[vi,vj] {求每个顶点的度数}
        end;
        total_d:=total_d+degree[vi];      {求总的度数}
        if odd(degree[vi]) then
          begin
            odd_num:=odd_num+1; {统计奇数顶点的个数}
            start:=vi          {确认从奇数顶点出发}
          end
    end;
  if odd_num>2 then writeln('no solution') {奇数顶点超过两个显示无解}
  else
    begin
      write('the road is: ',start);
      vi:=0;
      while total_d>2 do
      begin
```

```
            repeat vi:=vi+1 until graph[start,vi]<>0;{找连接的相邻点}
            if degree[vi]>1 then      {先画度数大于 1 的顶点}
            begin
                write('->',vi);
                graph[start,vi]:=0;
                graph[vi,start]:=0;
                degree[vi]:=degree[vi]-1;
                degree[start]:=degree[start]-1;
                total_d:=total_d-2;
                start:=vi;
                vi:=0
            end
        end;
        repeat vi:=vi+1 until graph[start,vi]<>0;      {确认最后一笔}
        writeln('->',vi)
    end
end.
```

输入图 1.3-6 所示的无向图，程序运行结果如下：

```
please input the number of vertex:6

please input the data:
```

0 1 1 0 0 0

1 0 1 1 0 1

1 1 0 0 1 1

0 1 0 0 1 1

0 0 1 1 0 1

0 1 1 1 1 0

the road is: 5->3->1->2->3->6->2->4->5->6->4

第 29 讲：算法概述

算法 Algorithm

算法是在有限步骤内求解某一问题所使用的一组定义明确的规则。通俗点说，就是计算机解题的过程。在这个过程中，无论是形成解题思路还是编写程序，都是在实施某种算法。前者是推理实现的算法，后者是操作实现的算法。

一个算法应该具有以下五个重要的特征：

有穷性： 一个算法必须保证执行有限步之后结束；

确切性： 算法的每一步骤必须有确切的定义；

输入：一个算法有 0 个或多个输入，以刻画运算对象的初始情况，所谓 0 个输入是指算法本身定义了初始条件；

输出：一个算法有一个或多个输出，以反映对输入数据加工后的结果。没有输出的算法是毫无意义的；

可行性： 算法原则上能够精确地运行，而且人们用笔和纸做有限次运算后即可完成。

备注： 算法可以没有输入。因为有些算法中包含了输入，如随机产生输入。

算法不一定必须在计算机上用某种语言实现。因为有些算法很难用某种编程语言来实现，例如人工智能

第 30 讲:排序算法

一、插入排序(Insertion Sort)

1. 基本思想：

每次将一个待排序的数据元素，插入到前面已经排好序的数列中的适当位置，使数列依然有序；直到待排序数据元素全部插入完为止。

2. 排序过程：

【示例】：

[初始关键字] [49] 38 65 97 76 13 27 49

J=2(38) [38 49] 65 97 76 13 27 49

J=3(65) [38 49 65] 97 76 13 27 49

J=4(97) [38 49 65 97] 76 13 27 49

J=5(76) [38 49 65 76 97] 13 27 49

J=6(13) [13 38 49 65 76 97] 27 49

J=7(27) [13 27 38 49 65 76 97] 49

J=8(49) [13 27 38 49 49 65 76 97]

```
Procedure InsertSort(Var R : FileType);
//对 R[1..N]按递增序进行插入排序, R[0]是监视哨//
Begin
for I := 2 To N Do //依次插入 R[2],...,R[n]//
begin
R[0] := R[I]; J := I - 1;
While R[0] < R[J] Do //查找 R[I]的插入位置//
begin
R[J+1] := R[J]; //将大于 R[I]的元素后移//
J := J - 1
end
R[J + 1] := R[0] ; //插入 R[I] //
```

end

End; //InsertSort //

二、选择排序

1. 基本思想：

每一趟从待排序的数据元素中选出最小（或最大）的一个元素，顺序放在已排好序的数列的最后，直到全部待排序的数据元素排完。

2. 排序过程：

【示例】：

初始关键字 [49 38 65 97 76 13 27 49]

第一趟排序后 13 ［38 65 97 76 49 27 49]

第二趟排序后 13 27 ［65 97 76 49 38 49]

第三趟排序后 13 27 38 [97 76 49 65 49]

第四趟排序后 13 27 38 49 [49 97 65 76]

第五趟排序后 13 27 38 49 49 [97 97 76]

第六趟排序后 13 27 38 49 49 76 [76 97]

第七趟排序后 13 27 38 49 49 76 76 [97]

最后排序结果 13 27 38 49 49 76 76 97

Procedure SelectSort(Var R : FileType); //对 R[1..N]进行直接选择排序

//

Begin

for I := 1 To N - 1 Do //做 N - 1 趟选择排序//

```
begin
K := I;
For J := I + 1 To N Do //在当前无序区 R[I..N]中选最小的元素 R[K]//
begin
If R[J] < R[K] Then K := J
end;
If K <> I Then //交换 R[I]和 R[K] //
begin Temp := R[I]; R[I] := R[K]; R[K] := Temp; end;
end
End; //SelectSort //
```

三、冒泡排序(BubbleSort)

1. 基本思想：

两两比较待排序数据元素的大小，发现两个数据元素的次序相反时即进行交换，直到没有反序的数据元素为止。

2. 排序过程：

设想被排序的数组 R［1..N］垂直竖立，将每个数据元素看作有重量的气泡，根据轻气泡不能在重气泡之下的原则，从下往上扫描数组 R，凡扫描到违反本原则的轻气泡，就使其向上"漂浮"，如此反复进行，直至最后任何两个气泡都是轻者在上，重者在下为止。

【示例】：

```
49 13 13 13 13 13 13 13
38 49 27 27 27 27 27 27
```

65 38 49 38 38 38 38 38

97 65 38 49 49 49 49 49

76 97 65 49 49 49 49 49

13 76 97 65 65 65 65 65

27 27 76 97 76 76 76 76

49 49 49 76 97 97 97 97

```
Procedure BubbleSort(Var R : FileType) //从下往上扫描的起泡排序//
Begin
For I := 1 To N-1 Do //做 N-1 趟排序//
begin
NoSwap := True; //置未排序的标志//
For J := N - 1 DownTo 1 Do //从底部往上扫描//
begin
If R[J+1]< R[J] Then //交换元素//
begin
Temp := R[J+1]; R[J+1 := R[J]; R[J] := Temp;
NoSwap := False
end;
end;
If NoSwap Then Return//本趟排序中未发生交换，则终止算法//
end
End; //BubbleSort//
```

四、快速排序（Quick Sort）

1. 基本思想：

在当前无序区 R[1..H]中任取一个数据元素作为比较的"基准"(不妨记为 X)，用此基准将当前无序区划分为左右两个较小的无序区：R[1..I-1]和 R[I+1..H]，且左边的无序子区中数据元素均小于等于基准元素，右边的无序子区中数据元素均大于等于基准元素，而基准 X 则位于最终排序的位置上，即 R[1..I-1]≤X.Key≤R[I+1..H](1≤I≤H)，当 R[1..I-1]和 R[I+1..H]均非空时，分别对它们进行上述的划分过程，直至所有无序子区中的数据元素均已排序为止。

2. 排序过程：

【示例】：

初始关键字 [49 38 65 97 76 13 27 49］

第一次交换后 ［27 38 65 97 76 13 49 49］

第二次交换后 ［27 38 49 97 76 13 65 49］

J 向左扫描，位置不变，第三次交换后 ［27 38 13 97 76 49 65 49］

I 向右扫描，位置不变，第四次交换后 ［27 38 13 49 76 97 65 49］

J 向左扫描 ［27 38 13 49 76 97 65 49］

（一次划分过程）

初始关键字 ［49 38 65 97 76 13 27 49］

一趟排序之后 ［27 38 13］ 49 ［76 97 65 49］

二趟排序之后 ［13］ 27 ［38］ 49 ［49 65］76 ［97］

三趟排序之后 13 27 38 49 49 ［65］76 97

最后的排序结果 13 27 38 49 49 65 76 97

各趟排序之后的状态

```
Procedure Parttion(Var R : FileType; L, H : Integer; Var I : Integer);
//对无序区 R[1,H]做划分，I 给以出本次划分后已被定位的基准元素的位置 //
Begin
I := 1; J := H; X := R[I] ;//初始化，X 为基准//
Repeat
While (R[J] >= X) And (I < J) Do
begin
J := J - 1 //从右向左扫描，查找第 1 个小于 X 的元素//
If I < J Then //已找到 R[J] 〈X//
begin
R[I] := R[J]; //相当于交换 R[I]和 R[J]//
I := I + 1
end;
While (R[I] <= X) And (I < J) Do
I := I + 1 //从左向右扫描，查找第 1 个大于 X 的元素///
end;
If I < J Then //已找到 R[I] > X //
begin R[J] := R[I]; //相当于交换 R[I]和 R[J]//
```

```
J := J - 1
end
Until I = J;
R[I] := X //基准 X 已被最终定位//
End; //Parttion //

Procedure QuickSort(Var R :FileType; S,T: Integer); //对 R[S..T]快速排序//
Begin
If S < T Then //当 R[S..T]为空或只有一个元素是无需排序//
begin
Partion(R, S, T, I); //对 R[S..T]做划分//
QuickSort(R, S, I-1);//递归处理左区间 R[S,I-1]//
QuickSort(R, I+1,T);//递归处理右区间 R[I+1..T] //
end;
End; //QuickSort//
```

五、堆排序(Heap Sort)

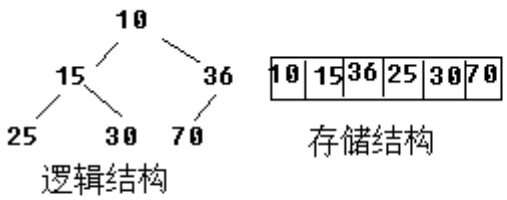
1. 基本思想：

堆排序是一树形选择排序，在排序过程中，将 R[1..N]看成是一颗完全二叉树的顺序存储结构，利用完全二叉树中双亲结点和孩子结点之间的内在关系来选择最小的元素。

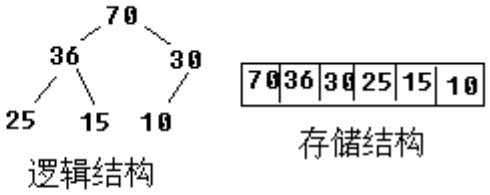
2. 堆的定义: N 个元素的序列 K1,K2,K3,...,Kn.称为堆，当且仅当该序

列满足特性：

Ki≤K2i Ki ≤K2i+1(1≤ I≤ [N/2])



小根堆示例



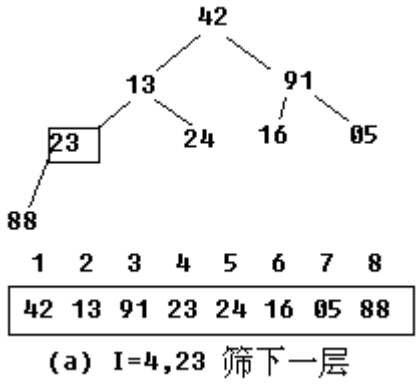
大根堆示例

堆实质上是满足如下性质的完全二叉树：树中任一非叶子结点的关键字均大于等于其孩子结点的关键字。例如序列 10,15,56,25,30,70 就是一个堆，它对应的完全二叉树如上图所示。这种堆中根结点（称为堆顶）的关键字最小，我们把它称为小根堆。反之，若完全二叉树中任一非叶子结点的关键字均大于等于其孩子的关键字，则称之为大根堆。

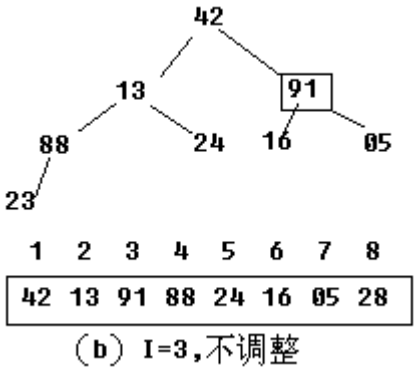
3. 排序过程：

堆排序正是利用小根堆（或大根堆）来选取当前无序区中关键字小（或最大）的记录实现排序的。我们不妨利用大根堆来排序。每一趟排序的基本操作是：将当前无序区调整为一个大根堆，选取关键字最大的堆顶记录，将它和无序区中的最后一个记录交换。这样，正好和直接选择排序相反，有序区是在原记录区的尾部形成并逐步向前扩大到整个记录区。

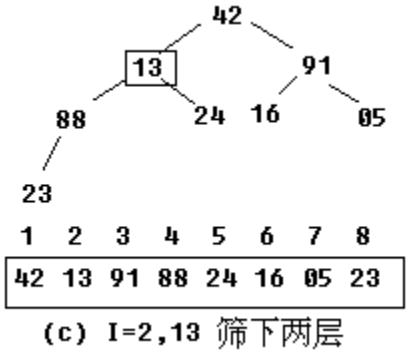
【示例】：对关键字序列 42，13，91，23，24，16，05，88 建堆



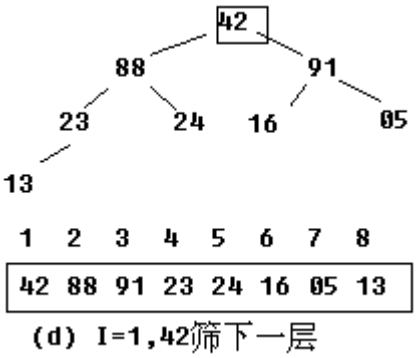
(a) I=4,23 筛下一层



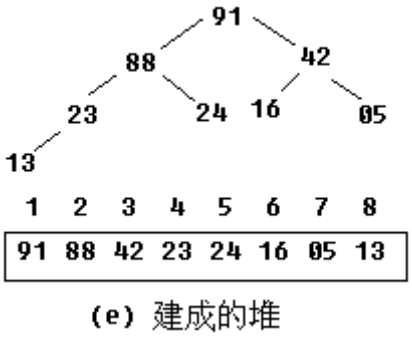
（b）I=3,不调整



(c) I=2,13 筛下两层



(d) I=1,42筛下一层



(e) 建成的堆

Procedure Sift(Var R :FileType; I, M : Integer);

//在数组 R[I..M]中调用 R[I]，使得以它为完全二叉树构成堆。事先已知其左、右子树(2I+1 <=M 时)均是堆//

Begin

X := R[I]; J := 2*I; //若 J <=M, R[J]是 R[I]的左孩子//

While J <= M Do //若当前被调整结点 R[I]有左孩子 R[J]//

```
begin
If (J < M) And R[J].Key < R[J+1].Key Then
J := J + 1 //令 J 指向关键字较大的右孩子//
//J 指向 R[I]的左、右孩子中关键字较大者//
If X.Key < R[J].Key Then //孩子结点关键字较大//
begin
R[I] := R[J]; //将 R[J]换到双亲位置上//
I := J ; J := 2*I //继续以 R[J]为当前被调整结点往下层调整//
end;
Else
Exit//调整完毕，退出循环//
end
R[I] := X;//将最初被调整的结点放入正确位置//
End;//Sift//

Procedure HeapSort(Var R : FileType); //对 R[1..N]进行堆排序//
    Begin
      For I := N Div Downto 1 Do //建立初始堆//
        Sift(R, I , N)
      For I := N Downto 2 do //进行 N-1 趟排序//
        begin
          T := R[1]; R[1] := R[I]; R[I] := T;//将当前堆顶记录和堆中最后一个记录交换//
```

```
            Sift(R, 1, I-1) //将 R[1..I-1]重成堆//
        end
    End; //HeapSort//
```

六、几种排序算法的比较和选择

1. 选取排序方法需要考虑的因素：

(1) 待排序的元素数目 n；

(2) 元素本身信息量的大小；

(3) 关键字的结构及其分布情况；

(4) 语言工具的条件，辅助空间的大小等。

2. 小结：

(1) 若 n 较小(n <= 50)，则可以采用直接插入排序或直接选择排序。由于直接插入排序所需的记录移动操作较直接选择排序多，因而当记录本身信息量较大时，用直接选择排序较好。

(2) 若文件的初始状态已按关键字基本有序，则选用直接插入或冒泡排序为宜。

(3) 若 n 较大，则应采用时间复杂度为 O(nlog2n)的排序方法：快速排序、堆排序或归并排序。 快速排序是目前基于比较的内部排序法中被认为是最好的方法。

(4) 在基于比较排序方法中，每次比较两个关键字的大小之后，仅仅出现两种可能的转移，因此可以用一棵二叉树来描述比较判定过程，由此可以证明：当文件的 n 个关键字随机分布时，任何借助于"比较"的排序算法，至少需要 O(nlog2n)的时间。

(5) 当记录本身信息量较大时，为避免耗费大量时间移动记录，可以用链表作为存储结构。

第 31 讲：列举（枚举）法

列举（枚举）法

枚举（ enumerate）法是基于计算机运算速度快这一特性的一种使用非常普遍的思维方法。它是根据问题中的条件将可能的情况一一列举出来分析求解的方法。但有时一一列举出的情况数目很大，如果超过了我们所能忍受的范围则需要考虑如何去排除不合理的情况，尽可能减少列举的问题可能解的数目。

例：求一元三次方程的解。

有形如： ax3+bx2+cx+d=0 一元三次方程。给出该方程中各项的系数 (a， b， c， d 均为实数)，并约定该方程存在三个不同实根 (根的范围在 -100 至 100 之间)，且根与根之差的绝对值 >=1。要求由小到大依次在同一行上输出这三个实根。

这道题的解法很多，最为简洁的方法就是直接枚举可能的实根解。

```
var a,b,c,d:real;i:integer;
function f(x:real):real;
begin
    f:=a*x*x*x+b*x*x+c*x+d
end;
begin
    readln(a,b,c,d);
    for i:=-10000 to 10000 do          {枚举方程所有可能的解}
    if abs(f(i/100))<1e-4 then write(i/100:2:2,'  ');
end.
```

猴子选大王

题目：有 M 个猴子围成一圈，每个有一个编号，编号从 1 到 M。打算从中选出一个大王。经过协商，决定选大王的规则如下：从第一个开始，每隔 N 个，数到的猴子出圈，最后剩下来的就是大王。

要求：从键盘输入 M，N，编程计算哪一个编号的猴子成为大王。

```
program tt;
const n1=100;
var a:array [1..n1] of integer;
m,n,i,j,k:integer;
begin
write('input m,n:');readln(m,n);
```

```
for i:=1 to m do a[i]:=i;
i:=1; k:=0;
while i<m do
begin
j:=0;
while j<=n do
begin
j:=j+1;
k:=k+1;
if k>m then
begin
k:=1;
while a[k]=0 do k:=k+1;
end;
end;
a[k]:=0;
i:=i+1;
end;
i:=1;
while a[i]=0 do i:=i+1;
writeln('The king is ',i);
end.
```

变形猴子选大王

题目：有 M 个人围成一圈，每人有一个编号，从编号为 1 的人开始，每隔 N 个出圈，按出圈次序排成一列，其编号刚好按顺序从 1 到 M。要求：从键盘输入 M，N，编程计算并输出这 M 个人原来在圈中的位置。

求数串的原始排列

题目：前 N 个自然数排成一串: X1,X2,X3.....Xn，先取出 x1,将 x2,x3 移到数串尾,再取出 x4,将 x4,x6 移到数串尾,..... 类推直至取完．
取出的序列恰好是:1,2,3.....n．要求输入 N,求原来的数串的排列方式.

```
var a:array[1..1000] of word;
n,i,j,dep:word;
begin
write('N(1-1000)=');
readln(n);
if (n=0) or (n>1000) then begin
writeln('Input error.');
readln;
halt;
end;
fillchar(a,sizeof(a),0);
```

```
a[1]:=1;
dep:=1;
for i:=2 to n do begin
j:=3;
while (j>0) do begin
dep:=dep mod n+1;
if a[dep]=0 then dec(j);
end;
a[dep]:=i;
end;
for i:=1 to n do write(a[i]:5);
writeln;
readln;
end.
```

狐狸捉兔子

问题：围绕着山顶有１０个洞，狐狸要吃兔子，兔子说：“可以，但必须找到我，?我就藏身于这十个洞中，你从１０号洞出发，先到1号洞找，第二次隔1个洞找，第三次隔2个洞找，以后如此类推，次数不限。”但狐狸从早到晚进进出出了１０００次，仍没有找到兔子。问兔子究竟藏在哪个洞里？

【答案】２，４，７，９

【参考程序 1 】

```
const
  holenumber=10;
var
  hole:array[0..holenumber] of 0..1;
  step,i,number:longint;
begin
  for i:=0 to 9  do hole[i]:=0;
  number:=0;
  for step:=1 to 1000 do begin
     number:=number+step; {循环地数}
     i:=number mod holenumber; {第几个洞}
     hole[i]:=1;
  end;
  for i:=0 to holenumber-1 do
     if hole[i]=0 then write(i:3);
  readln;
end.
```

【参考程序 2 】

```
const
  m=10;  {洞数}
```

```
var
   l,f,t:integer;
   a:array[1..m+1] of 0..1;
begin
  for t:=1 to m do
    a[t]:=0;
  f:=0; t:=0; l:=0;
  repeat
      l:=l+1;  {隔几个洞}
      t:=t+l;  {第几个洞}
    if t>m+1 then t:=t mod m;
    if t=0 then t:=10;{循环地找}
    a[t]:=1;{该洞已进过}
    f:=f+1; {进洞的次数}
  until f=1000;
  for t:=1 to 10 do
    if a[t]=0 then write(t,' ');
  readln
end.
```

第 32 讲：递推算法

递推算法

递归的很多问题可以转为递推（来处理，通常递推处理的效率比递归高得多。比如象阶乘、 Fibonacci 数列等。它们的相邻数之间有着明显的规律性的变化，通常可以将递归结束的条件作为递推的初始条件，并利用这种规律性一步一步递推到结果。这种递推通常采用循环迭代的方法，如循环累乘、循环累加等。

如递归中的例 1 转为递推算法时用循环累加来实现。

```
var f0,f1,f2:real;
    i,n:byte;
begin
    readln(n);
    f0:=1;f1:=2;
    for i:=2 to n do
    begin
        f2:=f0+f1;
        f0:=f1;
        f1:=f2
    end;
```

```
    writeln(f2:1:0)
end.
```

在用递归算法时，只要输入的 n 值稍大，程序求解就很困难，而递推则效率高很多。

在计算递归算法中的例 3 时，如果将自然数 n 的范围扩大到 1500 以内，则用递归算法递归调用的次数过多，在求 800 以上的数的时候就会出现困难，但用递推却可以大大缩小问题的规模。

递归算法中例 3 的递推程序：

```
var s:array[1..1500] of real;
    i,j,n:integer;
begin
    readln(n);
    for i:=1 to n do s[i]:=1;
    for i:=2 to n do
    if odd(i) then s[i]:=s[i-1]
    else
      for j:=1 to i div 2 do
          s[i]:=s[i]+s[j];
    writeln(s[n]:2:0)
end.
```

第 33 讲:搜索算法

搜索算法基础教程

大榕树

搜索算法是利用计算机的高性能来有目的的穷举一个问题的部分或所有的可能情况，从而求出问题的解
的一种方法。搜索过程实际上是根据初始条件和扩展规则构造一棵解答树并寻找符合目标状态的节点的过程。
所有的搜索算法从其最终的算法实现上来看，都可以划分成两个部分——控制结构和产生系统，而所有的算
法的优化和改进主要都是通过修改其控制结构来完成的。现在主要对其控制结构进行讨论，因此对其产生系
统作如下约定：

```
Function
ExpendNode(Situation:Tsituation;ExpendWayNo:Integer):TSitua
tion;
```

表示对给出的节点状态 Sitution 采用第 ExpendWayNo 种扩展规则进行扩展，并且返回扩展后的状态。
（本文所采用的算法描述语言为类 Pascal。）

一、回溯算法

回溯算法是所有搜索算法中最为基本的一种算法，其采用了一种“走不通就掉头”思想作为其控制结构，

其相当于采用了先根遍历的方法来构造解答树，可用于找解或所有解以及最优解。具体的算法描述如下：

[非递归算法]

```
<Type>
   Node(节点类型)＝Record
     Situtation:TSituation（当前节点状态）;
     Way-NO:Integer（已使用过的扩展规则的数目）;
   End
<Var>
   List(回溯表):Array[1..Max(最大深度)] of Node;
   pos(当前扩展节点编号):Integer;
<Init>
   List<-0;
   pos<-1;
   List[1].Situation<-初始状态;
<Main Program>
While (pos>0(有路可走)) and ([未达到目标]) do
Begin
  If pos>=Max then (数据溢出,跳出主程序);
  List[pos].Way-NO:=List[pos].Way-No+1;
```

```
  If (List[pos].Way-NO<=TotalExpendMethod) then (如果还有没
用过的扩展规则)
  Begin
    If (可以使用当前扩展规则) then
    Begin
      (用第 way 条规则扩展当前节点)
      List[pos+1].Situation:=ExpendNode(List[pos].Situation,

                                          List[pos].Way-NO);
      List[pos+1].Way-NO:=0;
      pos:=pos+1;
    End-If;
  End-If
  Else Begin
         pos:=pos-1;
       End-Else
End-While;
```

[递归算法]

```
Procedure BackTrack(Situation:TSituation;deepth:Integer);
Var  I    :Integer;
Begin
  If deepth>Max then (空间达到极限,跳出本过程);
```

```
  If Situation=Target then (找到目标);
  For I:=1 to TotalExpendMethod do
  Begin
    BackTrack(ExpendNode(Situation,I),deepth+1);
  End-For;
End;
```

范例：一个 M*M 的棋盘上某一点上有一个马，要求寻找一条从这一点出发不重复的跳完棋盘上所有的点的路线。

评价：回溯算法对空间的消耗较少，当其与分枝定界法一起使用时，对于所求解在解答树中层次较深的问题

有较好的效果。但应避免在后继节点可能与前继节点相同的问题中使用，以免产生循环。

二、深度搜索与广度搜索

深度搜索与广度搜索的控制结构和产生系统很相似，唯一的区别在于对扩展节点选取上。由于其保留了

所有的前继节点，所以在产生后继节点时可以去掉一部分重复的节点，从而提高了搜索效率。这两种算法每

次都扩展一个节点的所有子节点，而不同的是，深度搜索下一次扩展的是本次扩展出来的子节点中的一个，

而广度搜索扩展的则是本次扩展的节点的兄弟节点。在具体实现上为了提高效率,所以采用了不同的数据结构.

[广度搜索]

<Type>

Node(节点类型)＝Record

Situtation:TSituation（当前节点状态）;

Level:Integer(当前节点深度);

Last :Integer(父节点);

End

<Var>

List(节点表):Array[1..Max(最多节点数)] of Node(节点类型);

open(总节点数):Integer;

close(待扩展节点编号):Integer;

New-S:TSituation;(新节点)

<Init>

List<-0;

open<-1;

close<-0;

List[1].Situation<- 初始状态;

List[1].Level:=1;

List[1].Last:=0;

<Main Program>

While (close<open(还有未扩展节点)) and

(open<Max(空间未用完)) and

```
        (未找到目标节点) do
    Begin
        close:=close+1;
        For I:=1 to TotalExpendMethod do（扩展一层子节点）
        Begin
          New-S:=ExpendNode(List[close].Situation,I);
          If Not (New-S in List) then
          (扩展出的节点从未出现过)
          Begin
            open:=open+1;
            List[open].Situation:=New-S;
            List[open].Level:=List[close].Level+1;
            List[open].Last:=close;
          End-If
        End-For;
    End-While;
```

[深度搜索]

```
<Var>
    Open:Array[1..Max] of Node;(待扩展节点表)
    Close:Array[1..Max] of Node;(已扩展节点表)
    openL,closeL:Integer;(表的长度)
    New-S:Tsituation;(新状态)
```

```
<Init>
   Open<-0; Close<-0;
   OpenL<-1;CloseL<-0;
   Open[1].Situation<- 初始状态;
   Open[1].Level<-1;
   Open[1].Last<-0;
<Main Program>
   While (openL>0) and (closeL<Max) and (openL<Max) do
   Begin
     closeL:=closeL+1;
     Close[closeL]:=Open[openL];
     openL:=openL-1;
     For I:=1 to TotalExpendMethod do（扩展一层子节点）
     Begin
       New-S:=ExpendNode(Close[closeL].Situation,I);
       If Not (New-S in List) then
       (扩展出的节点从未出现过)
       Begin
         openL:=openL+1;
         Open[openL].Situation:=New-S;
         Open[openL].Level:=Close[closeL].Level+1;
         Open[openL].Last:=closeL;
```

```
        End-If
      End-For;
    End;
```

范例：迷宫问题，求解最短路径和可通路径。

评价：广度搜索是求解最优解的一种较好的方法，在后面将会对其进行进一步的优化。而深度搜索多用于只

要求解，并且解答树中的重复节点较多并且重复较难判断时使用，但往往可以用分支定界或回溯算法代替。

基本算法讲座 之 数学篇（1）

基本算法是解难题的基础，必须熟练掌握。这一讲将介绍跟数学密切相关的基本算法。（１）素数

数学类的基本算法大多数属于初等数论范畴，相当大一部分与素数有直接关系，因此素数是一个很基本又很重要的内容。

我们先来看看怎么判断一个数是否素数。素数的定义为：如果一个数的正因子只有 1 和这个数本身，那么这个数就是素数。根据定义，我们立即能得到判断一个数Ｎ（大于１)是否素数的简单的算法：枚举 2 到Ｎ－１之间的整数，判断是否能整除Ｎ。

如果 n 很大，那么上面的程序就要运行比较长的一段时间，那么

有没有更快一点的算法呢？回答是肯定的！因为如果 n 含有不为 1 和自身的因子，那么这些因子中必定有不大于 sqrt(n)的（假设 n 有因子 p，1<p<n，如果 p<=sqrt(n)，那么 p 就不大于 sqrt(n)，如果 p>sqrt(n)，那么 n/p 也是 n 的因子，而且 1<n/p<n，所以 n/p 不大于 sqrt(n)）。于是我们可以改进上面的程序，得到另外一个 Pascal 程序。容易知道这个算法的时间复杂度为 O(sqrt(n))。

（２）因式分解

因式分解的算法很简单，模拟手工分解的过程，我们得到分解 n 的算法：枚举所有不大于 n 的所有素数，判断这些素数能整除 n 多少次。判断 2 到 n 是否素数，总共要计算 sqrt(2)+sqrt(3)+sqrt(4)…+sqrt(n)<=n*sqrt(n)次，因此算法的时间复杂度可以粗略地认为是 O(n*sqrt(n))。事实上，我们有更好的算法。先看一个显而易见的结论：如果 p 是能整除 n 的所有大于 1 的数中最小的，那么 p 是 n 的一个素因子。基于这样一个结论，我们可以得到 Pascal 代码。

（３）公因子的数量

问题描述：已知一个正整数 N，问这个数有多少正公因子。

算法分析：最容易想到的算法是：枚举 1..N，看看有多少个数能整除 N，这种算法的复杂度为 O(N)。可以优化一下：如果 N 有小于 SQRT(N)的因子 X，那么 N 必定有大于 SQRT(N)的因子 Y 与 X 对应，而

且 XY=N。所以我们只需要枚举 1..SQRT(N)的数即可，还要考虑 N 为完全平方数的特殊情况。程序：Pascal。上面这个算法的复杂度为 O(sqrt(N))。其实我们可以利用因式分解的方法来做。假设我们已经分解 N 得到 N =(p[1]^s[1])*(p[2]^s[2])...*(p[pnum]^s[pnum])，其中 p[i] 为互不相同的素数，那么 N 的正因子的数量为（具体怎么推导请参考组合数学教材中的母函数一章）：(s[1]+1)*(s[2]+1)*…*(s[pnum]+1)。

（４）最大公因式

问题描述：已知两个正整数 a 和 b，求这两个数的最大公因数 GCD(a , b)。

(GCD 是 Greatest Common Divisor 的缩写)

算法分析：不妨设 a<=b，一种十分容易想到的算法是：枚举 1 到 a 的所有整数，在能同时整除 a 和 b 的数中取最大的。这个算法的时间复杂度为 O(min(a,b))，当 min(a,b)较大的时候程序要执行比较长的时间。我们可以利用初等数论中的一个定理：

GCD(a , b) = GCD(a , b-a) = GCD(a , b-2*a) = GCD(a , b-3*a) = … = GCD(a , b mod a)

关于这个定理的具体证明，请参考初等数论书（或者初中数学竞赛中的数论相关章节）。

下面给出利用这个定理来写的一个求最大公因式的程序，请读者仔细研究：Pascal。此算法的时间复杂度为 O(log(Max(a,b)))。（推导过程

请参考算法与数据结构教材）

（５）最小公倍数

问题描述：已知两个正整数 a 和 b，求这两个数的最小公倍数 LCM (a , b)。

(LCM 是 Least Common Multiply 的缩写)

算法分析：直接利用公式：LCM (a , b) * GCD(a , b) = a * b 即可。